Recommending Colorado Hiking Trails

Tyson O'Leary, Jackson Volesky

Introduction

Hiking is a popular sport around Colorado, and the world, but it can require an intimate knowledge of one's surroundings to get the full benefit. We seek to remove this barrier of entry by recommending to users hiking trails that suit them based on past hikes they have enjoyed. Aside from suggesting hikes based on their similarity to user input trails, our system could consider other factors such as weather, cell tower coverage, land cover, and reviews from other hikers. Hiking is an outdoor recreational activity thus the weather plays a key role in the overall experience. Cellphone coverage is also important in case any would be hikers were to get lost or injured. The engine we have built does not consider all these factors, its recommendations are based solely on similarity scores and current forecasts, but in the future including these factors would be an improvement. More generally than hiking trails, we are attempting to recommend geospatial locations based on user input, as well as qualitative and quantitative data respective to the location.

Our system is based on methods developed for the YouTube recommendation system [1]. Primarily we recommend trails through candidate generation, a content-based filtering system that narrows our data to a select few candidates, closely aligned with the user input data. The engine runs distributed across a cluster of computers as an Apache Spark job, and through a series of transformations computes similarity scores for all trails in the Colorado Trail Explorer dataset (containing over 96,000 datums [2]). The higher the similarity score, the closer a given trail is to matching the user input criteria. Once the similarity scores have been computed, we suggest the trails associated with the highest similarity scores and further rank them based on weather. Forecasts for each trail are obtained by contacting the National Weather Service's API [3].

We have worked extensively with geospatial data in the Sustain lab. Recommending subsets of geospatial locations, such as Census counties or tracts, has been a long-time interest of ours. The system we have built is but our first foray into geospatial focused recommendations. In future projects we will attempt to recommend location constraining queries based on past user queries and similarities between variables in the queries and other variables.

Problem Characterization

It is often difficult for hikers to find their next trail, especially for those lacking experience. There are many factors to consider when choosing a trail, some examples being a trail's length, change in elevation, trail terrain type, or rules about animals and vehicles. One would also consider weather information, including temperature, chance for precipitation, and wind speed. This information is constantly changing, which only makes choosing a trail more difficult. On top of all these considerations, the hiker must also take their distance from the trail into account. The best trails they find may be a long drive away.

We propose that this problem can be solved with a trail content-based recommendation system. This then presents the more academically interesting problem that we have solved. How can we synthesize the inherent and derived attributes of geospatial locations to accurately score and rank them as recommendations?

We are exploring this problem through the Colorado Trail Explorer (COTREX) dataset provided in collaboration by Colorado Parks and Wildlife and the Colorado Department of Natural Resources [2]. The dataset is highly voluminous, containing over 96,000 trails. Each individual feature composes forty trail qualitative and quantitative attributes, as well as the geometry of a trail presented as a Line String [4]. The size of the data makes this a difficult and time-consuming problem. We follow a distributed approach to ensure we can utilize high machine resources and parallelize our computations. We also take advantage of a distributed spatial computation framework.

Another challenge is data correctness. Empty values are prevalent in the data both in the attributes and the geometry of the trail. The empty values need to be dealt with carefully to ensure no information is lost. In the case of missing geometry, it does not make sense to recommend a hike that has no location, so these records can be set aside. The data has a particular coordinate reference system (CRS), which we need to ensure is correct for our operations. For example, if it is necessary to calculate centroids, a good CRS is EPSG:4326 [5].

Dominant Approaches

Calbimonte et al. [6] have built a recommendation system for hiking trails that is focused on the difficulty of trails based on several factors. They take advantage of semantics and relationships between trails and hiker ability levels to determine recommendation candidates. Their approach is similar to ours in that it is mostly content based; however, they have a more robust system that includes a mobile app for users to report obstacles and perceived difficulty levels on trails.

Vías et al. [7] define a system to propose routes in a protected area. Their system uses advanced techniques in network analysis, multi-criteria decision analysis, and GIS to recommend the routes that would be viable for hiking and safe for the environment. Our system is much more generalizable and does not suffer from their system's limited scope. Some ideas proposed to account for the safety of hikers could be incorporated into a future version of this recommendations system.

Existing Applications including the AllTrails mobile application and the COTREX web application [8], [9] provide tools for manual exploration. These tend to feature a manipulable map and filtering capabilities, which are certainly useful for finding trails. However, this puts all the responsibility on the user to understand attributes about trails and find what they are looking for. Our recommendations are more accessible because they do not require any knowledge from the user besides an input trail or two.

Covington et al. [1] present an expansive description of the recommendation engine that powers YouTube. It does not recommend trails, but it is a prominent recommendation engine. It is built on two deep neural networks: one for candidate generation and one for ranking candidates. The candidate generation network uses collaborative filtering, an approach that recommends based on similarity between users. Collaborative filtering would very likely improve the quality of our system's recommendations because it could capture the hidden attributes of hikes that cannot or are not expressed in public datasets.

Methodology

The majority of our analysis utilizes distributed operations and aggregations provided by Apache Spark [10]. The cluster is set up in standalone mode, with automatic configuration management scripts. The master node is configured to use five gigabytes of memory and four cores. Executors are started on a configurable number of machines.

Our methodology is split into three stages: preprocessing, candidate generation, and ranking. The latter two are derived from the stages presented in [1]. The preprocessing stage consists of sourcing the data and transforming downloaded files into analysis-ready formats. The goal of candidate generation is to narrow the full candidate space of trails down to those that are scored highly for recommendation. The final ranking stage takes the resultant suggested trails and sorts them based on factors that are unscored or more computationally expensive.

Preprocessing

The trails used in this project are sourced from the Colorado Trail Explorer (COTREX) dataset [2], which is provided in ESRI Shapefile format [11]. The full dataset is split into three sub-datasets containing all the trails, designated trails, and trailheads. We only focus on the collection that contains all the trails to ensure we have as many available as possible. The spark job requires that the input data is in GeoJSON format [4]. Apache Sedona does provide a built-in function for reading in shapefiles, but it was troublesome, and we were unsuccessful in getting it to work properly. Instead, we opted for the ogr2ogr command line tool provided by GDAL to convert from Shapefile to GeoJSON [12].

Before entering the meat of our computations, we must transform the trail geometry into Sedona geometry. This will enable us to use the entire library of spatial analysis functions provided by Sedona. These operations require that the geometry is in the EPSG:4326 coordinate reference system, which is equivalent to the world geodetic system of 1984 projection commonly used by GPS [5]. We found that the system becomes very fragile if there are any null values present for the geometry of any features. A trail with no location information is meaningless for us, so all null geometries are dropped before the Sedona conversion.

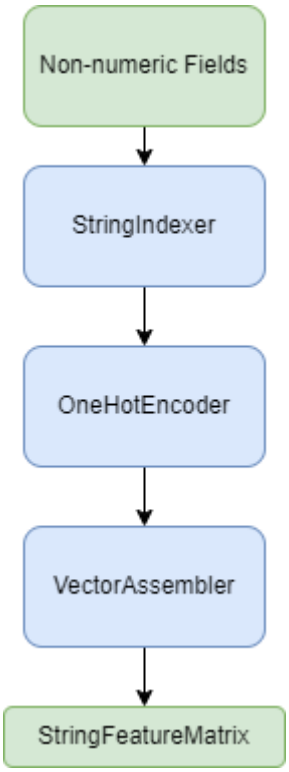
Candidate Generation

Our engine follows a content-based filtering approach to generate recommendation candidates. The goal is to transform all features into rows that can be scored and sorted to facilitate narrowing the search space down to the best candidates. Different types of attributes need to be handled differently. There are three major types of attributes: categorical, numeric, and geometric.

Categorical features are vectorized into a one-hot vector representation. Spark MLLib provides helpful operations to accomplish this. The first step is to pass all string columns into a StringIndexer, which assigns an index to each unique value in the column ordered by frequency. The OneHotEncoder then uses the generated indexing scheme to create a one-hot vector representing the value. The columns are finally passed through a VectorAssembler, which assembles the one-hot vectors into a full feature vector. This vector acts as a way to describe the trail categorically with a numerically manipulable value.

The similarity score of a feature is then calculated based on user inputs. The user provides string-valued names of trails that they have hiked and enjoyed in the past. After features have been vectorized, the data frame is filtered for the rows that match the inputs. It is important to locate these matched rows after vectorization is complete to ensure the generated feature vectors have a matching schema. If the matching

rows were instead located early and vectorized separately, the StringIndexer would likely assign indexes in a different order.



The found vectors are then merged into a single vector representing a virtual trail that has all the features of all the user's trails. The vectors are combined through vector addition, which has the side-effect of some elements being higher than one. These elements are those features that exist in more than one input trail vector. For example, if two trails both allow dogs, the element of the combined feature vector associated with the allowance of dogs would equal two. Our similarity metric is the dot product. In the case that both vectors are one-hot, the dot product is equivalent to counting the number of ones in the same position in both vectors, which is a logical metric in the case of similarity because a pair of trails should be more similar than another pair if they have more attributes in common. The inclusion of greater values in the combined input vector means that the attributes common to all inputs will contribute more to the similarity score.

The next metric utilizes the numeric fields. These fields are the length of the trail, the starting elevation, and the final elevation. A trail is more similar to the user's trails as the difference between the average length of a user trail and the length of the given trail decreases. Similar logic applies to the trail's elevation gain, calculated from the difference between the maximum and minimum elevation.

Our final recommendation metric is the distance to the trail from the user's current location. We utilize built-in Sedona operations to accomplish this. The centroid of every trail is first calculated to get a single-point reference for the trail's location, which is used in the distance calculation. The centroid is also used later in the ranking stage. Then, the distance is calculated to the centroid using Sedona's haversine function implementation. There should not be a large performance difference added by calculating the centroid and distance in succession because Sedona would calculate the centroid before the distance anyway [13].

With calculations for similarity score, length difference, elevation gain difference, and distance finished, the system can choose the final candidates. We follow a progressive narrowing approach, which narrows using the metrics ordered same as was just described.

1. The top one thousand most similar trails are taken.
2. The one hundred trails with the lowest difference in length are taken.
3. The fifty trails with the lowest difference in elevation gain are taken.
4. The final generated candidates are the ten trails closest to the user.

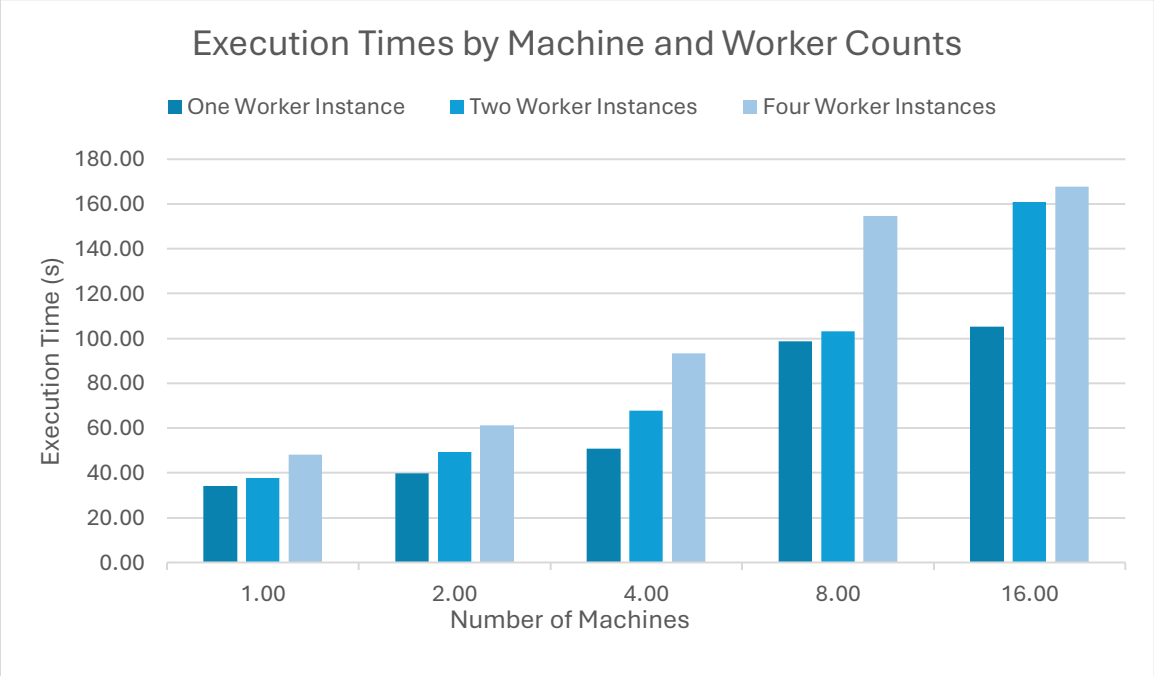
Ranking

The final ranking stage takes the ten generated candidates from the previous stage and reorders them based on current conditions. The spark application is wrapped in a python-based command line interface to ease the development load for user input operations and API connections. The wrapper starts the spark job, awaits its completion, and reads the resulting candidates. It then issues a query to the weather API [3] that includes the centroid of each candidate to find the weather at each trail at the moment of execution. The candidates are hierarchically sorted by temperature, wind speed, and chance for precipitation. This is a major benefit of our system because it allows each recommendation to be dynamic based on when the user requests recommendations.

Experimental Benchmarks

Considering our lack of a test set and how subjective hiking trail recommendations are, we focused our quantitative analysis on execution time. We first ran the program with different combinations of machines and worker instances. In this experiment we used a desired temperature of 60°F; for input trails we used Horsetooth Rock and Arthur's Rock hiking trails. We only timed each combination of machines and worker instances once. Effects on execution time are assessed using linear regression at α=0.05. An increase in either the number of machines running the program, or the number of worker instances on each machine, is associated with an increase in execution time. Time to run increases by 7.10 seconds for each additional machine (p-value=1.78 x 10^{-6}), and by 12.75 seconds for each additional worker instance (p-value=0.00413). Increasing the number of worker instances per machine affects execution time almost twice as much as adding another worker machine does (e.g., eight machines with four workers each should perform worse than 16 machines with two worker instances per machine).

Number of Machines	Number of Worker Instances	Execution Time (s)
1	1	34.29
1	2	37.79
1	4	48.18
2	1	39.74
2	2	49.30
2	4	61.33
4	1	50.95
4	2	67.82
4	4	93.49
8	1	98.80
8	2	103.04
8	4	154.63
16	1	105.20
16	2	160.80
16	4	167.81



In our next experiment, the number of machines and worker instances were fixed to 10 and one respectively. We ran the program in five different configurations three times, each with a larger set of inputs than the next. Trails were selected from the Lory State Park area and reused in each set of inputs. I.e., the trail from the first trial was used in the second trial, both trails from the second trial were used in the third, and so on. Running

the program with an increasing number of input hiking trails resulted in no significant increases in execution time (p-value=0.835).

Number of Input Trails	Average Execution Time (s)
1	54.74
2	59.86
4	58.93
8	58.89
16	56.53

The Spark job requires the most compute time, followed by fetching the forecasts for the ten recommended trails and then ranking those trails based on their similarity score and current forecast. With the number of machines and worker instances still fixed at 10 and one, we timed the execution of multiple sections of the program five times and recorded the averages.

Program Section	Average Execution Time (s)
Spark Job	64.62
Query NWS API	0.8434
Rank Trails	0.003988
Total	65.59

Almost the entire execution time (98.5%) is spent running the spark job, where we compute the similarity scores and return the ten most similar trails. In the remaining time, the trails are ranked based on their current forecast, which is retrieved from the National Weather Service API.

Insights Gleamed

This project was challenging in that it required us to process data in a non-serialized manner. The distributed nature of Apache Spark required our logic to conform to the new paradigm. This was especially difficult when working with the spark SQL data frames as it began to feel more like a database querying problem. The data frames were necessary to use the built-in functions provided by MLLib, which aided our processing during content-based filtering. It is not immediately obvious how to compute similarity scores between trails using a large quantity of qualitative attributes. Following the methodology outlined in [1], we simplified the process by vectorizing the qualitative attributes. This presented a simple calculation for the similarity score that used vector dot product. After building the engine, it is apparent that this solution provides a reasonable similarity score because of properties associated with dot products. It is equivalent to counting the number of common ones between two binary vectors, which is exactly the logic we would want to use when comparing qualitative attributes. If both trail surfaces are dirt, then they have that attribute in common and therefore are similar in that dimension.

The other important insight is related to our benchmarks. After examination of the runtimes, it is obvious that this problem did not require distribution to be efficient. Our job runs fastest when run sequentially on one worker. This implies either that our data is too small for distribution to make a difference, our transformations are too wide, or we are improperly loading and distributing the data. It is also possible that the problem would become better suited for distribution if we grew the data to more than just trails from

Colorado. We did not use HDFS to back our file storage, which may have affected our distribution. Instead, we attempted to leverage the computer science department's network file system; however, we are unsure whether the data was properly distributed. In the future, we could use preprocessing techniques in tandem with a file system more amiable to distributed data processing to reduce the total amount of computations that must be performed every execution, as well as reduce the number of computations each individual worker performs.

How will the problem space look in the future?

Trails are already popular among the public. As more people adopt trail information applications, they will become increasingly similar to social media. This is already evident in existing applications, such as AllTrails and other exercise tracking apps. With high user interaction comes high volumes of data, enabling more advanced filtering approaches, such as collaborative filtering. Especially intriguing is a location-based recommendation system [14] that could be fine-tuned to recommend the best trails near a user. Following this paradigm would evolve the recommendations to be more personalized and allow the system to account for local and global user preferences. If hikers agree that a certain trail is unenjoyable, the system will learn to recommend that trail infrequently or not at all. If a certain user does not like a trail, then the system would know not to recommend it to that user or any users it deems similar.

The freedom and creativity of this system could emerge in the final ranking stage. Candidate generation would no longer rely on the attributes of the trail, and therefore would become more reliable and scalable, thus the number of trails that could be included would grow immensely. The attributes describing a trail would be free for use in direct filtering and final ranking operations. The final ranking could also be determined by outside attributes like the day's weather (as described), cell tower coverage, land cover type, or animal patterns. It is already possible to incorporate these in our system, but it would become brittle.

Conclusions

The engine that we have built successfully suggests hikes based on user input trails and desired weather conditions. Whether or not these recommendations are *good* recommendations is hard to say. Providing the system a desired temperature of 60°F, Horsetooth Rock and Arthur's Rock trails, and Fort Collins as the user's location produces a list of the ten similar hikes. These hikes are the Herrington Trail, Wild Loop Trail, Black Powder Trail, Mt. McConnel Trail, a NIST Service Road, Little Raven Trail, Lake Agnes Trail, Burro Spur to Windy Peak Trail, and Horseshoe Trail. The first five trails listed are all within an hour's drive of Fort Collins and, excepting Horseshoe Trail, the rest are within two hours of the input user location. Horseshoe trail is located in the southwest corner of Colorado. The tenth trail recommended by the system bore no name, something that our system will have to manage in the future. Overall, our engine provided seven strong suggestions for hikes that the user may enjoy. We consider the NIST Service Road (located in Boulder, CO), Horseshoe Trail, and the unnamed trail to be poor recommendations. Either because they are not a hike, too far to be easily accessible, or because they have no identifier. It does not serve to recommend trails that are not trails or are unlocatable because of their lack of an identifier. In the future, we will have to constrain our data to named hikes.

Aside from recommendations, our system does not excel at running in a distributed fashion. The fastest execution time we saw was when running the program on one machine with one worker instance, although

running the program on two machines was comparable. One cause could be that our data is not being sufficiently divided between the nodes. We believe that each node is looking at the entire dataset, instead of a fraction of it. By better dividing our search space in the future, we posit that we would see large improvements in execution time. Another cause of slow execution time could stem from the specific distributed manner we run the program in. Using Spark Sedona requires that we compile a "fat .jar" (a .jar file which contains all necessary dependencies as well as the source code) which then needs to be distributed to the entire cluster. The fat .jar is hundreds of megabytes in size, thus distributing it to every node is an I/O intensive process. If we used an alternative that did not require the fat .jar, we could possibly reduce execution time.

Bibliography

[1] P. Covington, J. Adams, and E. Sargin, "Deep Neural Networks for YouTube Recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*, Boston Massachusetts USA: ACM, Sep. 2016, pp. 191–198. doi: 10.1145/2959100.2959190.

[2] The Colorado State Trails Program, "Colorado Trail Explorer (COTREX)." Colorado Information Marketplace. Accessed: Apr. 28, 2024. [ESRI Shapefile]. Available: https://data.colorado.gov/Recreation/Colorado-Trail-Explorer-COTREX-/tsn8-y22x/about_data

[3] N. US Department of Commerce, "National Weather Service API Web Service." Accessed: Apr. 28, 2024. [Online]. Available: https://www.weather.gov/documentation/services-web-api

[4] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub, and S. Hagen, "The GeoJSON Format." Internet Engineering Task Force, Aug. 2016. doi: 10.17487/RFC7946.

[5] Klokan Technologies (last), "WGS 84 - World Geodetic System 1984, used in GPS - EPSG:4326." Accessed: Apr. 28, 2024. [Online]. Available: https://epsg.io

[6] J.-P. Calbimonte, S. Martin, D. Calvaresi, and A. Cotting, "A Platform for Difficulty Assessment and Recommendation of Hiking Trails," in *Information and Communication Technologies in Tourism 2021*, W. Wörndl, C. Koo, and J. L. Stienmetz, Eds., Cham: Springer International Publishing, 2021, pp. 109–122. doi: 10.1007/978-3-030-65785-7_9.

[7] J. Vías, J. Rolland, M. L. Gómez, C. Ocaña, and A. Luque, "Recommendation system to determine suitable and viable hiking routes: a prototype application in Sierra de las Nieves Nature Reserve (southern Spain)," *J Geogr Syst*, vol. 20, no. 3, pp. 275–294, Jul. 2018, doi: 10.1007/s10109-018-0271-8.

[8] "AllTrails." AllTrails. Accessed: Apr. 28, 2024. [Online]. Available: https://www.alltrails.com/mobile

[9] "Colorado Trail Explorer (COTREX)." Accessed: Apr. 28, 2024. [Online]. Available: https://trails.colorado.gov/

[10] M. Zaharia *et al.*, "Apache Spark: a unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016, doi: 10.1145/2934664.

[11] ESRI, “ESRI Shapefile Technical Description”. Available: https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/whitepapers/pdfs/shapefile.pdf

[12] E. Rouault *et al.*, “GDAL.” Open Source Geospatial Foundation, Apr. 04, 2024. doi: 10.5281/ZENODO.5884351.

[13] Apache Sedona™, “Function Documentation.” Accessed: Apr. 28, 2024. [Online]. Available: https://sedona.apache.org/latest/api/sql/Function/#st_distancesphere

[14] J. Bao and Y. Zheng, “Location-Based Recommendation Systems,” in *Encyclopedia of GIS*, S. Shekhar, H. Xiong, and X. Zhou, Eds., Cham: Springer International Publishing, 2017, pp. 1145–1153. doi: 10.1007/978-3-319-17885-1_1580.