Lost Person Study

Garret Carver, Kameron Lightheart,
Chris Coca, Tosh Wilcox

July 10, 2020

Abstract

We perform unsupervised analysis on topography files covering a region of the Rocky Mountains in Utah and Colorado to determine if it can shed any insight upon the features that the topography contains. We also perform supervised analysis upon the output of a lost person simulator to see if the method can learn the behavior of the simulation given a topography input. While our unsupervised analysis methods mainly did not perform as intended, we found somewhat meaningful results from our supervised analysis. More work and perhaps a different type of dataset will be needed if this study is to be continued.

1 Problem Statement

Getting lost in the woods is an increasingly uncommon phenomenon but when it does occur the results are often tragic. The demographic of lost persons has evolved over time but today it is mostly the mentally ill, children, and the very old. Therefore, determining a reasonable search area as quickly as possible and understanding what terrain is possible to travel over is very important in minimizing search time. Common sense and the result of various studies tell us that the faster a person is found the higher the chance of survival.

Research in this area has been done almost exclusively by Dr.Koester who has been working since the 1980s on collecting and analyzing incident reports by Search and Rescue organizations all over the world. His published works include constructing probability heat maps and a Bayesian approach to finding the most likely location of a lost person. Using his large database he has also been able to do the statistical work that explains the differences in behavior of children, the autistic, and people with dementia.

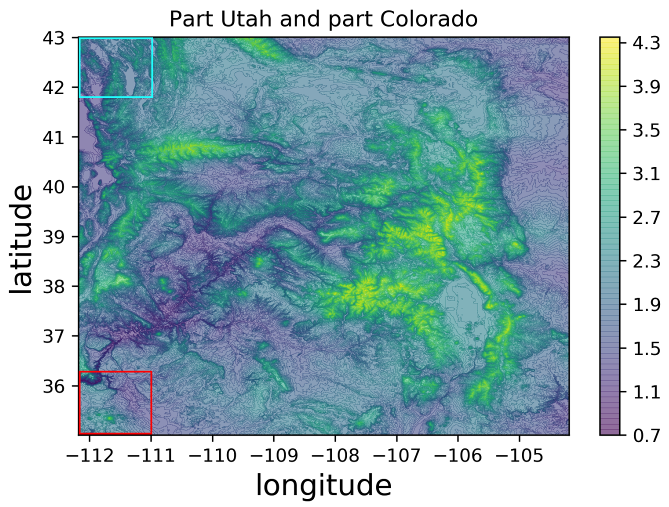
Last semester Garret attempted to get this dataset from Dr. Koester but was unable due to the fact that it is proprietary. In place of this he created a simulator that would aggregate the behavior of many random walks that were motivated to walk downhill. By dividing up the topography into zones, starting the random walks in the center of each zone, and by keeping track of where each walker ended up, Garret was able to create an adjacency matrix that showed how connected each zone was to its neighbors. The “connectedness” was determined by the proportion of random walkers that ended up going from zone a to zone b. This adjacency matrix could then be used as a way to interpret how ’travelable’ a piece of terrain is.

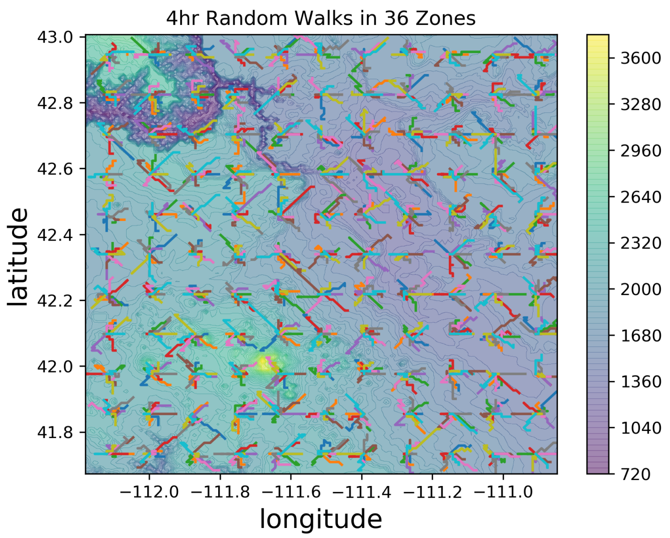
This project is split into two parts, unsupervised learning and supervised learning. Our goal for unsupervised learning it to analyze the topography file with various methods such as PCA, NMF, RPA, etc to see if any of these methods prove useful at separating out the different features in the terrain. We perform similar analysis on the output of the lost person simulator. The goal for supervised learning is to use the topography file as an input to a model that will then attempt to learn the output of the lost person simulator that was written last semester.

2 Data

There are several open source databases that contain all kinds of topographic data and we got ours from edex-cloud.unidata.ucar.edu, a database maintained by the Advanced Weather Interactive Processing System (AWIPS). We were wary of downloading LIDAR data or any of the other seemingly exotic types because we did not want to do a lot of cleaning, however this fear was unfounded as topography data is fairly standardized. Great care should be taken to make sure that the indices of the topography file are correctly matched with their latitude and longitude coordinates. This is something that we did not originally make much of an effort to do and it ended up costing us a lot of time.

Last semester Garret Carver created a program that would accept a topography file, divide it up into grids and then aggregate the behavior of many different random walks that were biased to walk downhill. This was done with the intention of simulating the behavior of a lost person moving over a given topography. To make the program more lightweight the original topography file was divided up into 36 nearly equal zones that were then analyzed using his method. The output of the method produces an "adjacency matrix" that encodes the proportion of random walkers that ended up in each zone.





This simulation has several obvious problems, the first being that the behavior of a lost person is more complicated than simply walking down hill. Additionally, the 36 "window" files have incorrect latitude coordinates. Above, we see in the original figure that the blue box (top left corner) corresponds with the latitude/longitude displayed but the actual topography data comes from the red box (bottom left corner). This was ignored due to the fact that the time limit for procuring a dataset had already been surpassed and it does not really affect what we are trying to do. Another problem is that the adjacency output of the simulation is very sparse. Below we show a table that describes the sparsity of each of our datasets and we will explain in further detail what information is being encoded.

`2020-03-28_19.20_4_5_10_100`	`Sparcity 0.97`
`2020-03-28_19.29_8_5_10_100`	`Sparcity 0.94`
`2020-03-28_19.52_16_5_10_100`	`Sparcity 0.97`
`2020-03-28_14.20_32_5_10_100`	`Sparcity 0.97`

The file is named as such: `date.trial_time_cuts_subcuts_100`. Given in more detail, the second number after the "." is the simulated walk time, after that is the number of cuts to the original topography. For each dataset we used 5 cuts which gives 6 pieces in each axis so 36 'windows' total. The next number corresponds to the number of cuts within each window and we defaulted to 10 cuts which gives 11 pieces in each axis so 121 pieces total. This results in an adjacency matrix that is 121x121. If we had used every point then the adjacency matrix would be 25921x25921.

The adjacency matrix encodes the proportion of random walkers that traveled to each zone, where each row sums to one. An entry along the diagonal represents the proportion of how many random walkers ended up back into the zone that they started from and every other entry in the row corresponds to the proportion of random walkers that went from zone i to zone j. In the table above, the number after "Sparcity" is the the percentage of walkers that ended in the zone in which they started or in other words, the "sparse percentage" of the corresponding adjacency matrix. Hence we are skeptical of our roughly diagonal adjacency matrices because we expected them to be less sparse.

2.1 Ethical Considerations

The topography data is legal to use and provided for free by AWIPS, a government agency. Since the dataset we are working with is entirely generated by a model that simulates human behavior, there is little to consider with ethics. However, use of this model in a practical situation would not be wise. As mentioned in the problem statement, we wanted to get actual lost person data but quickly realized that there is not an easy way to find standardized missing person data. Consequently, Garret built a program to model lost person behavior and we used this to simulate the path of travel a lost person would make given different starting points. Using this data, we made predictions on where a lost person would end up after a certain amount of time. Since the predictions are based upon simulated data, which has not been validated, the practical use of this information for finding a lost person is not advised.

On the other hand, if we did have data on missing people such as path of travel and survival rate, then we would have some interesting questions to consider. If we can figure out which parts of a terrain people are most likely to get lost in or which parts are especially dangerous, do we then have an ethical responsibility to make those terrains unavailable? If so, do we have the right to prevent people from entering those areas? It's also worth noting that if we chose to train/validate a model using real missing person data, we would only have training data from a subset of that population. There is another subset of missing people that we can't include in our training or testing data because searchers never found their bodies. Can we rely on a model that was trained/validated on an incomplete subset of the population of lost people? This reminds us of the problem that Abraham Wald solved of minimizing damage to bomber planes by adding armor to places where the surviving planes didn't have bullet holes. Thus, if we decided our goal was to minimize death then are we learning anything if our data mainly consists of behaviors and paths of travel of missing people who survived? Would the optimal policy actually be to close the areas where missing people didn't travel through? These are all questions that will need to be handled carefully in future work.

3 Methods: Unsupervised Analysis

3.1 Principle Component Analysis

Our topography data consists of 3 features, longitude, lattitude and altitude. Since our features are all measured in the same units, we realized that we could use linear techniques of Principal Component Analysis (PCA) to not only reduce the dimension of our data, but also reduce the temporal complexity of other algorithms that use this data. Since a surface can be described with just two parameters, we performed the PCA transform with 2 components and we were able to capture 85% of the original variance. However, the components and the plot below are not easily interpreted. This motivated our attempts to implement Nonnegative Matrix Factorization afterward.

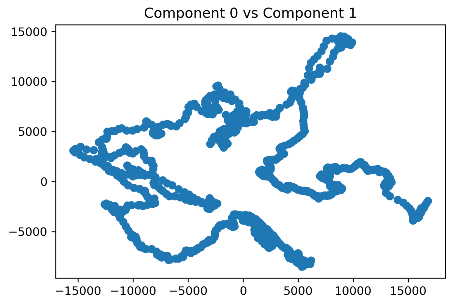
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Create PCA Model
pca = PCA(n_components=2, svd_solver='full')

# Fit model on topography
topography = np.load('OG_topo,npy')
pca.fit(topography)

# Transform data
new = pca.transform(topography)

# Plot components
plt.scatter(new[:,0], new[:,1])
```



Although it is difficult to extrapolate meaning from the above plot, we felt confident that this method would be useful for reducing complexity. To move forward with supervised machine learning methods, we decided to perform PCA using 4 components which contains 95% of the variance of the data. This reduces the number of columns of our data to 4.

3.2 Nonnegative Matrix Factorization

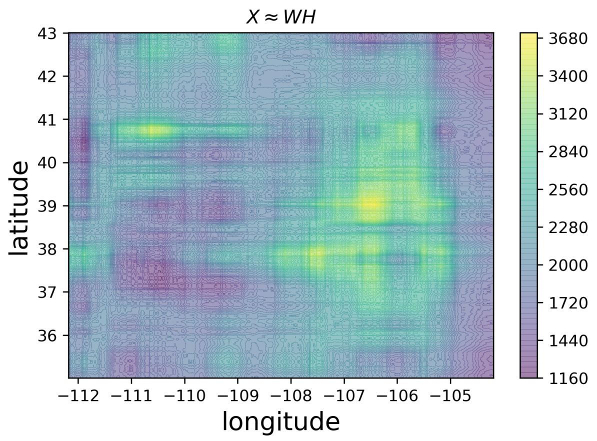
The main goal of Nonnegative Matrix Factorization (NMF) is to reduce the number of data points that must be recorded and since NMF splits data into nonnegative matrices, the factorization usually becomes much easier to interpret. For this reason we spent some time using NMF in hopes of classifying where certain landforms are such as mountains, rivers, lakes, cliffs, etc. This would give us a wholistic approach to analyzing the terrain rivaling our simulation. In other words, if we know where certain landforms are before we generate our random walkers, then we may be able to predict the outcome of their path with greater accuracy.

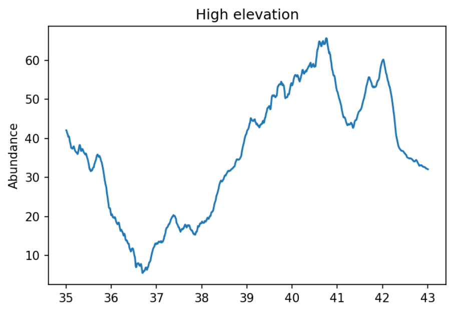
At first, we used 4 components to hopefully classify 4 landforms but, similar to what happened with our Princpal Component Analysis, we had a hard time understanding what the components meant. After many attempts we realized that we were treating our data as if it was a hypersectral image. This obviously doesn't make sense since our data only consists of elevation, not wavelengths. Instead of using many components, we decided to choose 2 corresponding to high and low elevations. By looking at the plot below and comparing it with the original plot on page 2, we can clearly see that green corresponds to high elevation and purple corresponds to low elevation.

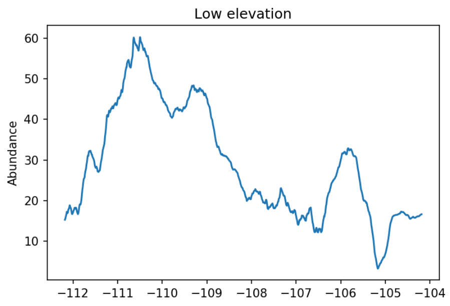
```
from sklearn.decomposition import NMF

# Get data
X = topography

# Apply NMF using 2 components
model = NMF(n_components = 2, init = ‘random’, random_state = 0)
W = model.fit_transform(X)
H = model.components_
```







Above, the left graph plots the first component against latitude and the right graph plots the second component against longitude. We realized that interchanging the x-axis with latitude and longitude did not change the shape of the graph and for this reason, plotting the components in 3D gave no new meaning. If the components are plotted against longitude and are compared with the NMF plot and the original plot, one could say that the first component corresponds with high elevation and the second component corresponds with low elevation; we titled them as such.

However we were still at a loss; we were unable to confidently derive meaning from the plots because the shape of the graph does not change when we interchange the x-axis with latitude and longitude. Thus, we decided that we may be better off by directly classifying parts of the data as “Mountain” or “Not Mountain” by simply giving a threshold to the altitude. For example, we could say that anything above an altitude of 2800 could be considered a mountain and anything lower than that is not a mountain.

In the end, we decided to move forward without NMF application. As mentioned earlier, the main reason that NMF did not work as we intended is because we were treating our data as if it were a hyperspectral image. That gave us the idea of applying NMF to a screenshot of the topography using google earth. However, since the screenshot is in RGB format, it only made sense to take 3 components and this did not give us any new information either.

3.3 Random Dimension Reduction

Random Dimension Reduction not only reduces dimension, but it also preserves distances between points in a dataset. It appears that for our given input of 960 vectors of 960 altitudes, we can only reduce our dimensionality at a cost of distorting our data by about 25%. Thus it is not in our best interest to conduct a RDR according to the bound given by the Johnson Lindenstrauss Theorem. We will run a randomized dimension reduction anyways to compare with PCA.

```
from sklearn.random_projection import johnson_lindenstrauss_min_dim

def preserves_distance(X, A, eps):
    """
    Checks if the matrix A preserves the relative distance between all pairs of points in X

    Parameters:
        X (() ndarray) Original data set
        A (() ndarray) Modified data set
        eps (float) percentage of distortion allowed
    Returns:
        is_acceptable (bool) Boolean value indicating whether the matrix X preserves the
        relative distance between all pairs of points woth a distortion of no more than eps
    """

    m,n = X.shape
    for i in range(m):
        for j in range(i+1, m):
            lower = (1 - eps) * np.linalg.norm(X[i] - X[j])**2
            upper = (1 + eps) * np.linalg.norm(X[i] - X[j])**2
            middle = np.linalg.norm(A @ X[i] - A @ X[j])**2

            if lower > middle or upper < middle:
                return False
    return True
```

```
k = 210 dimensions, eps = 0.7, took 1 iterations before passing
k = 329 dimensions, eps = 0.5, took 1 iterations before passing
k = 468 dimensions, eps = 0.4, took 1 iterations before passing
k = 762 dimensions, eps = 0.3, took 1 iterations before passing
k = 983 dimensions, eps = 0.26, took 1 iterations before passing
```

Thus, we see that for a few different values of epsilon, a random projection can be found quite quickly. In this case, it was found on the first try, but that might not always be the case. RDR appears, in our testing with the simulation, to be inferior to PCA due to the fact that only about 75% of the variance is being captured with this method.

4 Methods: Supervised Analysis

4.1 Data Formatting

Supervised machine learning is used to create a model to learn about how each input is mapped to the output so that the results can be reproduced and predicted. For our problem the input is a topology matrix and the output is an adjacency matrix. To be able to use any of the machine learning tools we have learned about we had to simplify the data. To do this, we first did PCA with 4 components on each topology file to reduce the number of columns to 4. Next we flattened each matrix and concatenated them together to get one input matrix with all of the condensed topology information. For the adjacency matrix we found the indices of the most likely spots that the simulated lost person would end up in and appended these together to get a condensed output matrix.

4.2 New Method: MultiOutputClassifier

Due to the fact that for each data point our output is a vector of indices, we had to research a new method of doing machine learning. We found that sklearn has a MultiOutputClassifier function that let us use random forests, xgboost, and other machine learning methods. The MultiOutputClassifier combines with sklearns other machine learning functions to extend the fitting and predicting processes to a series of classifiers, instead of just one.

Because the output is a vector of indices instead of an integer or float, the way we measure success had to be evaluated. To get a measure of the accuracy for our predictions we compared each value in the vector to the actual value and found the percentage of how many we got exactly right. Then we took an average of all of those percentages.

$$f(\hat{y}_i) = \begin{cases} 1 & \text{if } \hat{y}_i = y_i \\ 0 & \text{otherwise} \end{cases}$$

$$\text{M} = \text{the number of predictions made}$$

$$\text{N} = \text{the number of indices in each prediction}$$

$$\text{accuracy} = \Bigg(\sum_{j=1}^{\text{M}} \Big(\sum_{i=1}^{\text{N}} f(\hat{y}_{j_i}) \Big) / \text{N} \Bigg) / \text{M}$$

4.3 Random Forest

A random forest is a model that is made up of of several decision trees. Each decision tree is generated with N randomly selected (with replacement) points from the data set. This process is called bagging and it ensures that the decision trees will not be correlated. Each individual tree will “vote” and the classifier with the most votes will be chosen by the random forest.

We used a randomized grid search to find approximately which parameters would work best, then we used a grid search to narrow our parameters search. With the parameters tuned, we trained and tested the model. We were able to achieve an accuracy of 38% which sounds bad but the random forest is trying to classify indices from the adjacency matrix, meaning that there are 121 classes for it to choose from.

```
from sklearn.multioutput import MultiOutputClassifier

# Use the MultiOutputClassifier with a random forrest classifier
rfc = MultiOutputClassifier(RandomForestClassifier(n_estimators=420, min_samples_leaf=14,
min_samples_split=5,max_features='sqrt', max_depth=None, bootstrap=True))

# fit the model
rfc.fit(X_train, y_train)

# get predictions
preds = rfc.predict(X_test)
print(test_accuracy(actual=y_test, predictions=preds))
```

```
The Accuracy for the Random Forest Classifier is 0.376
```

4.4 XGBoost

Extreme Gradient Boosting (XGBoost) regularizes the building of additional trees by expressing the objective in terms of a sum over the leaves of the trees. To use XGBoost we used the same training and testing data that was used to test the random forest. We did a grid search for the best parameters and then fit and tested the model. We got an accuracy score of 32%.

```
from sklearn.multioutput import MultiOutputClassifier

# Use the MultiOutputClassifier with a xgboost classifier
xgb1 = MultiOutputClassifier(xgb.XGBClassifier(max_depth=6, n_estimators=200))

# fit the model
xgb1.fit(X_train, y_train)

# get predictions
preds = xgb1.predict(X_test)
print(test_accuracy(actual=y_test, predictions=preds))
```

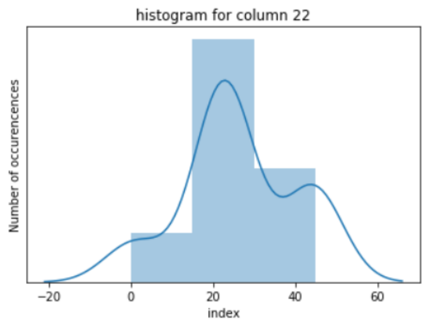
```
The Accuracy for the xgboost classifier is 0.323%
```

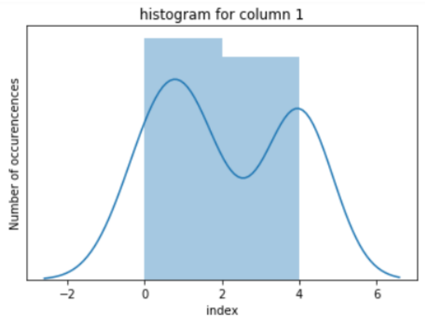
5 Results

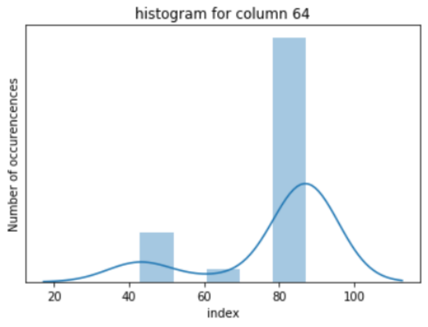
As mentioned earlier, we did not gain as much new information from our unsupervised analysis as we hoped for. However, we were still able to use PCA techniques to reduce the dimension and complexity of our data for the machine learning methods

It is difficult to know whether the accuracy scores we got form the random forest and xgboost are impressive or not. As mentioned earlier, the adjacency matrix is very sparse, perhaps the random forest and xgboost models are not really making predictions based on the topology and they are really just learning to predict indices along the diagonal of the adjacency matrix.

To test this we fit an empirical distribution for each column of the testing data set. Then we did a monte carlo simulation to sample from the each distribution to get a prediction. We did this 1000 times and got an average accuracy of 28%. This is 10% lower than the random forest which was the best predictive model that we used. Because this prediction was made independently of the topology data we know that the data that we are testing on, the adjacency matrix, is somewhat predictable. However, using the topology data in a random forest yields significantly better results. This means that the random forest learns about the landscape and makes its predictions somewhat based on the landscape. Below are histograms for a few of the columns of the monte carlo simulation prediction.







6 Analysis/Conclusions

6.1 Unsupervised Analysis

Our unsupervised analysis focused mainly on dimension reduction since there is no real ability to do any clustering. Our initial hopes were to use this analysis to classify or come up with labels for our topographical data. In the end, we decided that the aforementioned methods did not really give us new information. However, unsupervised analysis could prove to be extremely interesting if we could somehow acquire hyperspectral images of the area that our topographical data represents. In future work, we would love to be able to train on images and come up with a model that classifies landforms, which we believe would add a whole new meaning to our analysis.

6.2 Supervised Analysis

In our supervised analysis, we decided that the reason our accuracy scores are so low are because we don't have enough training data. The close proximity of the monte carlo accuracy and the random forest accuracy is likely due to the fact that we didn't have very much data. The training data set only had 25 rows. If we had more data then the models that we used would be able to learn more about the data and would hopefully get higher accuracy.