Remote Sensing Image Interpretation

Colorado School of Mines Field Session

Newmont Mining Team #2

Rebecca May, Justin Persinger, Paul Sattizahn

Client: Brian Krzys

June 22nd, 2017

Table of Contents

Introduction

Description of Client

Newmont Mining was founded in 1921 in Greenwood Village, Colorado. They are now one of the largest gold producers across the globe and the only one listed in the S&P 500 index. Their goal is to create value and improve lives through sustainable and responsible mining. Our client works with infrared satellite data to locate mineral deposits and find new potential mining sites.

Product Vision

Newmont Mining makes extensive use of remote sensing data while searching for mineral deposits. The collected data is stored as geo-tagged TIFF files consisting of nine Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) bands with precision of thirty meters. These bands hold infrared values which can be interpreted to determine the mineral composition of the Earth. Interpretation of the satellite imagery is often done manually, making it difficult to complete at a large scale in an efficient amount of time. The goal of this project was to develop a cloud-based machine learning application that automates this process and produces results equal to or better than the manual interpretation. The original vision was a web application with a map interface that allows a user to select a few known points and use them to generate a heatmap of locations that share similar infrared values and patterns.

Requirements

This project can be broken down into two segments: the web application and the machine learning aspect. Our client left many of the specifications for both parts up to us to decide. The specifications that we agreed upon are outlined below.

Functional Specifications

1. The web application should display a map that allows the user to create, edit, and delete named point sets.

2. Once a set of points has been created and all of the training points have been added to it, similar points on the map (within the region that data exists for) should be found.

3. The user should select which method is used for machine learning or numerical analysis. These methods are described under technical design in this document.

4. A heatmap of the generated results should be displayed and the user should be able to download it.

Non-Functional Specifications

1. The web application needs to interact with the Postgres database that is provided by the client. The purpose of the database is to save point sets that have been generated in the web application to allow the user to switch from one set to another within a session as well as storing submitted data between sessions. The points can then be retrieved from the database to be analyzed with machine learning.

2. The web pages must be served on the Apache web server provided by the client.

3. JavaScript should be used to handle web interaction.

4. Python should be used for the backend machine learning.

Features

The features of the web application, as well as ideas for expansion, are described in this section. Features of the rest of the project are described in the Technical Design section below.

Features of the Web Application

In Appendix A, see Figure 04 for a screenshot of the web application, Figure 05 for a screenshot of the web application with the results overlaid, and Figure 06 for a screenshot of the generated heatmap once downloaded.

1. Features of the map:
    a. Map data is overlaid on the Google Maps display.
    b. Users are only able to add points within the boundaries of existing data.
    c. The overlay is tiled so that the resolution of the overlay increases as a user zooms in on a specific area.
    d. Users can change the maps display between map and satellite view.
    e. Users can search specific locations through the use of a search bar.

2. Features of point sets: (see Figure 01)
    a. Users can select the point set to work on using a dropdown.
    b. Users can create, delete, and rename point sets.
    c. Users can select which method to use when comparing points.
    d. Users can choose to include neighboring points in the calculation.

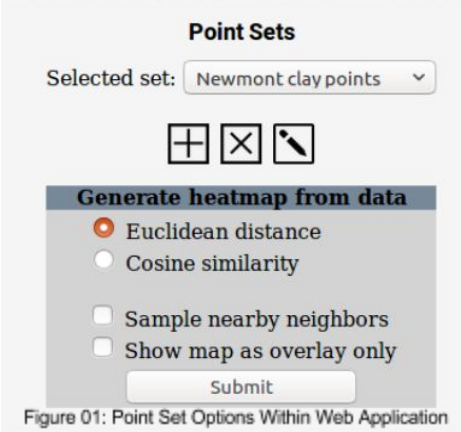

Figure 01: Point Set Options Within Web Application

3. Features of points:
    a. Once a point set is selected from the dropdown, all points within that set appear in the table.
    b. Users can add and delete points from a set.
    c. When users select a point in the table, the selected point bounces on the map, allowing the user to verify specific markers.

4. Once the submit button is pushed, a heatmap is generated as a TIFF file and the points from the point set are shown as green squares on the file. The user then has the option to:
    a. Have the results overlaid on the map in the browser.
    b. Download the generated TIFF file.

5. All buttons on the web application have tooltips which are shown when hovering over them to assist users.

Features Not Implemented

The project definition asked for the use of machine learning to identify similar areas, however, the requirements of the project evolved as the project progressed. Instead of machine learning, we are using the Euclidean distance and cosine similarity formulas to generate results. The user has the option to pick which method is used as well as if they would like the eight neighboring pixels to each selected point to be taken into consideration. All other requested features have been implemented. These methods are further described under Technical Design in this document.

Future Work

We have implemented the major features necessary to meet the specifications of our client. However, the following is a list of features that could be added.

1. Given a sufficient amount of training and tests points, implement a machine learning algorithm instead of (or in addition to) the two distance formulas.

2. Allow the user to select which of the nine ASTER bands are used.

3. Add user accounts so that users only interact with the data sets that they create.

4. Prompt the user to upload their own files to be used, allowing the web app to be used for any location where satellite data has been collected.

5. Add functionality to the search bar so that a user can type in latitude and longitude values to place a point.

6. Add a legend to the web application so that the user is informed of what percentiles each color of the generated heat map represents.

System Architecture

The architecture of the Newmont Remote Sensing Image Interpretation web application can be analyzed as two parts, backend and frontend. The backend is comprised of a Postgres database, Python code for data processing, and further data formatting using the Geospatial Data Abstraction Library (GDAL) tools. The frontend consists of an Apache web server using PHP and handles user interaction with JavaScript. A simple visualization of the architecture is depicted below in Figure 02.

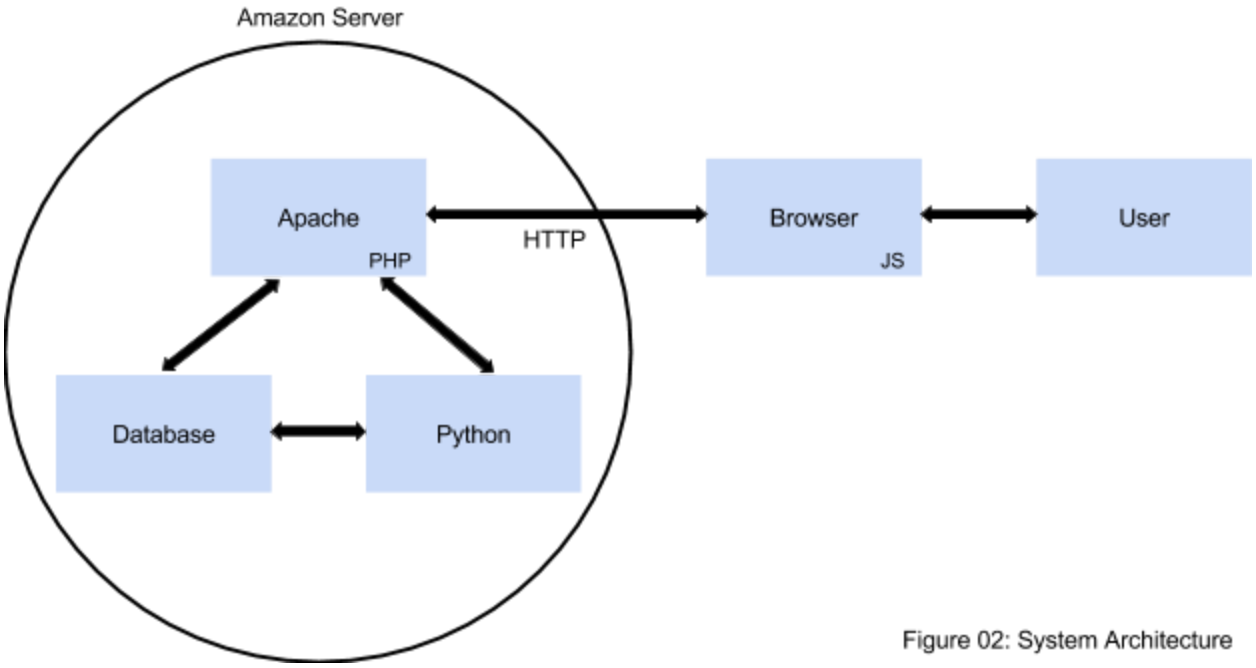

Figure 02: System Architecture

Backend

1. **Database (Postgres)**: The database is used to store the point sets and their associated points that the user creates in the web application. It is queried when the user modifies or switches between point sets and when the Python data processing code is called on a particular set to generate results.

2. **Data processing (Python)**: The Python code is used to perform calculations on the data based on input points. This includes performing distance metrics and various statistical analyses of the input data which are passed to GDAL for use in producing a heatmap for visual comparison.

3. **Data formatting (GDAL)**: In order to display the maps on the browser, GDAL2Tiles tool was used. This utility is further described under Technical Design in this document below. The GDALdem utility was used to read in a color file and adjust the color of each pixel on the heatmap to display specified percentiles as specific colors.

Frontend

1. **Apache web server & PHP**: The web server exposes a REST API, made in PHP, to allow interaction with the client. It parses data given through the API and interacts with the database accordingly, performing queries to match any specified actions. It can have varying responses, such as data the client requested and simple responses notifying the client that the query executed successfully. The web server delivers the web application to the client in HTML, CSS, and JavaScript. Lastly, the web server executes our Python code on the Amazon server in response to methods available on the API.

2. **User interaction (JavaScript)**: JavaScript handles all user interaction and interfaces with the REST API via AJAX in order to propagate changes the user makes and request various data. It listens to user actions on the webpage and manipulates the HTML Document Object Model (DOM) to sync data both ways between the website and the web server. This component also has several designs in place to allow for ease in further extension of app features and modification of existing ones.

Data Flow

An example flowchart of the flow of data through the application is provided in Figure 03 below. After the user creates a point set and populates it, they request generation of a heatmap. The Python code is run with the selected point set and distance metric to use in calculations. The Python code interacts with the database in order to retrieve the relevant points. It performs calculations on the geographical data, making comparisons between the points provided and the rest of the data. It stores the results as a geoTIFF file. Once the file is generated, Gdaldem colorizes the data based on specific percentile ranges. If the user selected to have the heatmap overlaid in the browser, Gdal2Tiles is used to generate tiles for the image. Otherwise, a download link for the file is generated. The generated image or link is sent to the web server which is then passed on to the browser and handled appropriately.

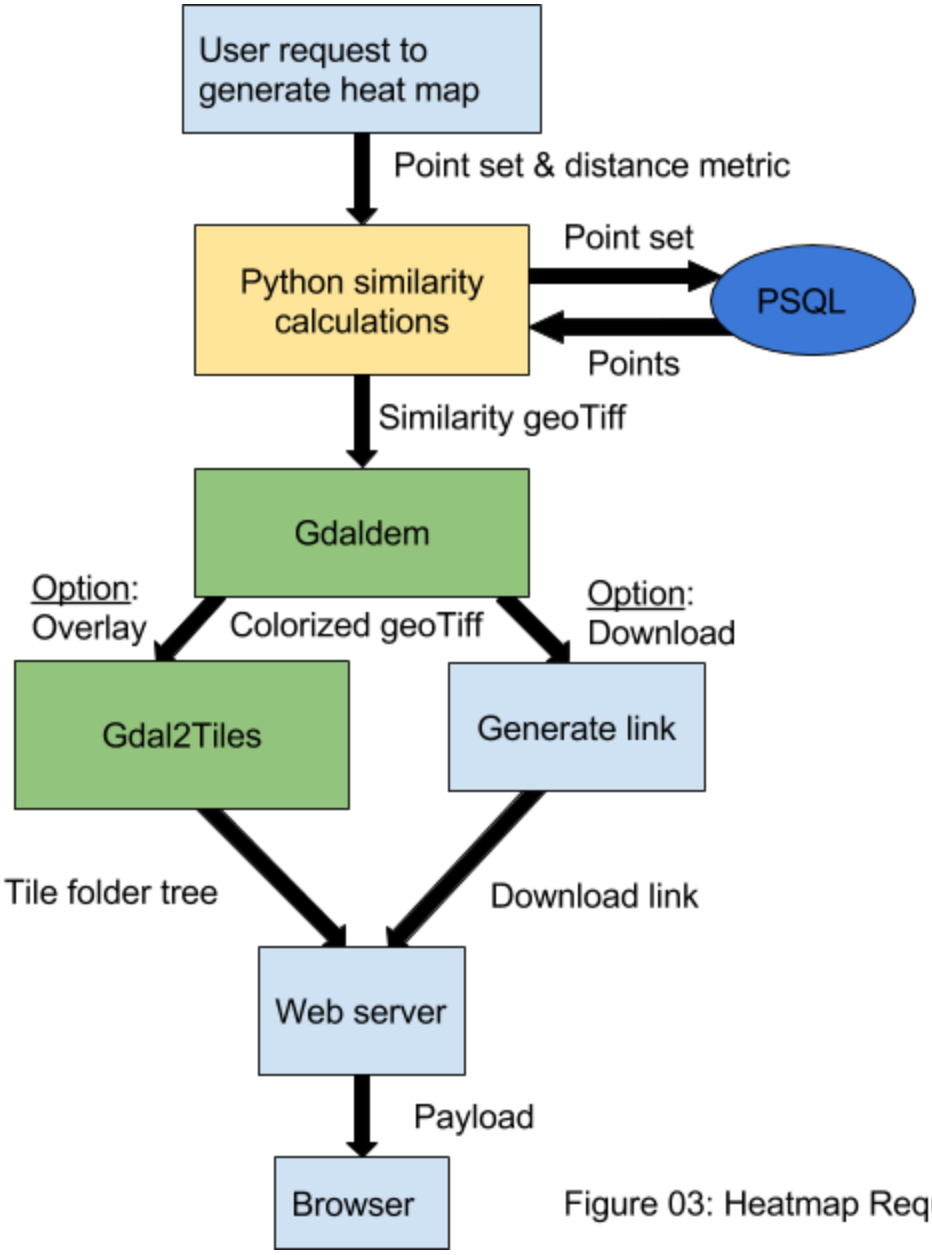

Figure 03: Heatmap Request Data Flow

Technical Design

This project required extensive knowledge of geography and geology as well as basic web development skills. However, we found the most interesting aspects of the final result to be the formulas used to find locations similar to the training points and the method of tiling to overlay our maps onto the standard Google Maps image.

Distance Formulas

The user of the web application is prompted to select from several options before their results are generated. The first is what formula they would like used to find similar points and then whether or not they would like to have nearby neighbors of those points selected included in the calculations.

Euclidian Distance

The Euclidian Distance Formula for the physical distance between two points p and q with n dimensions is calculated as $d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$. We have slightly adapted this formula to be able to apply it to our data.

Each point has nine layers associated with it. Each of these layers consists of a single value corresponding to the infrared data at that location. Before the Euclidian Distance formula can be used, the mean of the set of training points is calculated. The mean of the points is calculated for each layer, weighing them equally, resulting in an array with a length of nine being generated. We can call this one dimensional array of the means ‘means’ and the three-dimensional array of the rest of the image ‘img’. A new three dimensional array ‘differences’ is now generated and populated with the difference between each point in img and means, squared ($differences = (img - means)^2$). The result is then compressed into a two dimensional array by averaging the values present at each (x,y) coordinate. The array is inverted and returned as $1 - \sqrt{differences}$ so that the most similar points to the mean of the input have the highest value. The entire result is also multiplied by a constant to stretch the range of values returned in order to make the results more apparent.

Cosine Similarity

Cosine Similarity is most commonly used as a measure of the cosine angle between two non-zero vectors and it is calculated with a Euclidean dot product. The cosine similarity between vectors ‘a’ and ‘b’ is calculated as:

$$similarity = \sum_{i=1}^{n} A_i B_i \div \left(\sqrt{\sum_{i=1}^{n} Ai^2 \sum_{i=1}^{n} Bi^2}\right)$$

where A_i and B_i are components of vectors a and b respectively.

Once again, the mean of the training points was calculated at each layer. This time the values were put into a one dimensional vector called ‘means’ and the data for the entire image was put into a three dimensional vector called ‘img.’ The numerator of the equation above is stored into a variable called ‘top’ and is calculated with a call to the NumPy dot product function between img and means. The denominator is the product of the normalized version of img and means. We then divide the numerator by the denominator, take the arccosine of the result, and divide it by pi/2. These results are stored in a two dimensional array called ‘cos_sim.’ The results are then subtracted from one so that the points that are most similar have the highest value.

Sampling Nearby Neighbors

The two methods described above to locate points similar to those selected by the user rely on the user selecting the exact pixel that they intended. We know that this is unlikely to be accomplished. Therefore, we have added an option for the eight pixels surrounding the one the user selected to be considered in the calculations. Selecting this method is likely to increase the accuracy of the application because the number of training points increases by eight for each training point used, especially when intended for interpretation of geographical data.

Tiling of Map Overlays

One of our hopes was that the application would be efficient and that the user would not be left waiting for different aspects of the web application to load. However, this project dealt with a substantial amount of map data which needed to be transferred from the server to the browser in order to overlay the map of data into the web application. To make this more efficient, we have tiled the map data for specific zoom levels using GDAL2Tiles. This utility takes a map and generates a directory of smaller files for specific zoom levels. At the most zoomed out level allowed on the web application, the entire image is displayed as one file of lower resolution. As the user zooms in, smaller map segments with higher resolution are displayed. This reduces data transfers to be less wasteful as the highest resolution is not needed at the more distant zoom levels.

Decisions

As a team, we had the opportunity to make many decisions concerning the language, tools, and design as long as we were able to meet the specifications of our client. A summary of the decisions made are outlined below.

Language Choices

1. **Python:** The original vision of the project was to implement a machine learning algorithm which made Python appealing as many existing machine learning libraries are written in Python. Although we did not end up using machine learning, we still used Python for finding similar points as it had bindings for GDAL which made interaction with our geographical data much easier. We used the Python GDAL library for dealing with many of the geological aspects of the project.

2. **PHP:** The web server was built using PHP. Although there are many alternatives that could be considered, our client wanted the focus of our project to be on the machine learning results, not the web application or server. We chose PHP because it makes writing code to serve web pages trivial.

3. **JavaScript:** The web application was made with JavaScript (leveraging jQuery). This decision was natural as the Google Maps API uses JavaScript. jQuery also provides various functions for interacting with REST services, like our web server, while encoding and parsing data with little boilerplate.

Tools Used

1. **Google Maps API:** The Google Maps API was chosen for our map representation as it provides a lot of built-in functionality. We wanted to limit the amount of code we would have to write unrelated to the primary task of our project while still maintaining adaptability.

2. **PostgreSQL:** PostgreSQL was chosen for the server database as all team members are familiar with it and it is widely used. It is also the primary database used by our client.

3. **GDAL:** We used GDAL for interacting with all of the geographical data for our project. GDAL is the industry standard for low-level interaction with geographical data. Also, if we had spent time trying to write code to perform the same operations, we would have spent a lot of time sidetracked from our more important tasks.

Results

The web application meets the needs and specifications of the client. The following is a summary of how we achieved high levels of performance and usability while fulfilling those needs.

Performance Testing Results

There were very apparent improvements in performance when generating geoTIFF files for our results. The first time we tried writing our own method using Python in order to find the distance of each pixel from the mean of our training points, but running the code took anywhere from 13 to 23 minutes. We were able to make use of the NumPy library and cut the run time of this segment of code down to an average of two seconds. The website is also very responsive; the user does not need to wait for any feature on the page to load before use and the markers on the map update as soon as a new point set is selected. We have tiled the map that gets overlayed on the Google Map so that a smaller resolution photo is used while the user is zoomed out, allowing the image to be displayed in a more efficient manner.

Summary of Testing

Due to the lack of appropriately sized training and testing point sets, it is very difficult to objectively test the accuracy of the outputted heatmap, although we are able to test the results visually. We made several new point sets that included points from large grassy fields or bodies of water as these are easy points for non-geologists to identify. The generated results were easy to verify as accurate and showed promising results for identification of similar areas.

Results of Usability Tests

The website is fully functional and tested on Chrome, Firefox, Safari, IOS, and Android and the features are intuitive. Each button on the web application has tooltips on them to make sure the user understands their functionality. When selecting to delete a point set, the user is required to select a checkbox in a dialog box to confirm that they know they will not be able to retrieve the current data once it has been deleted.

Appendices

A. Images of Web Application

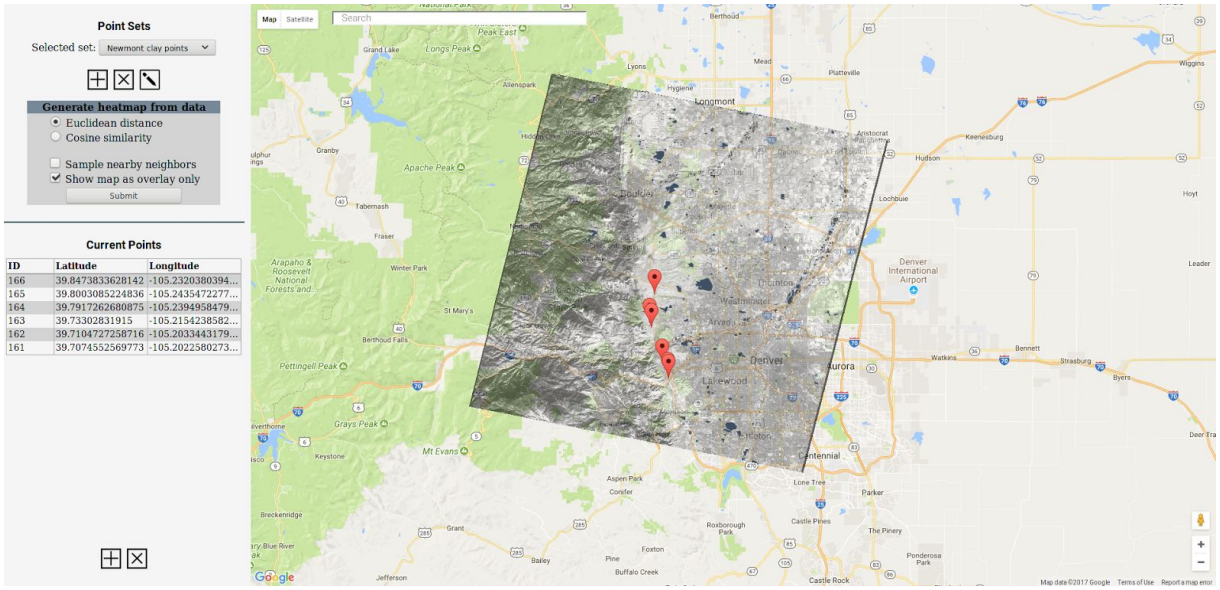

Figure 04: Screenshot of the web application

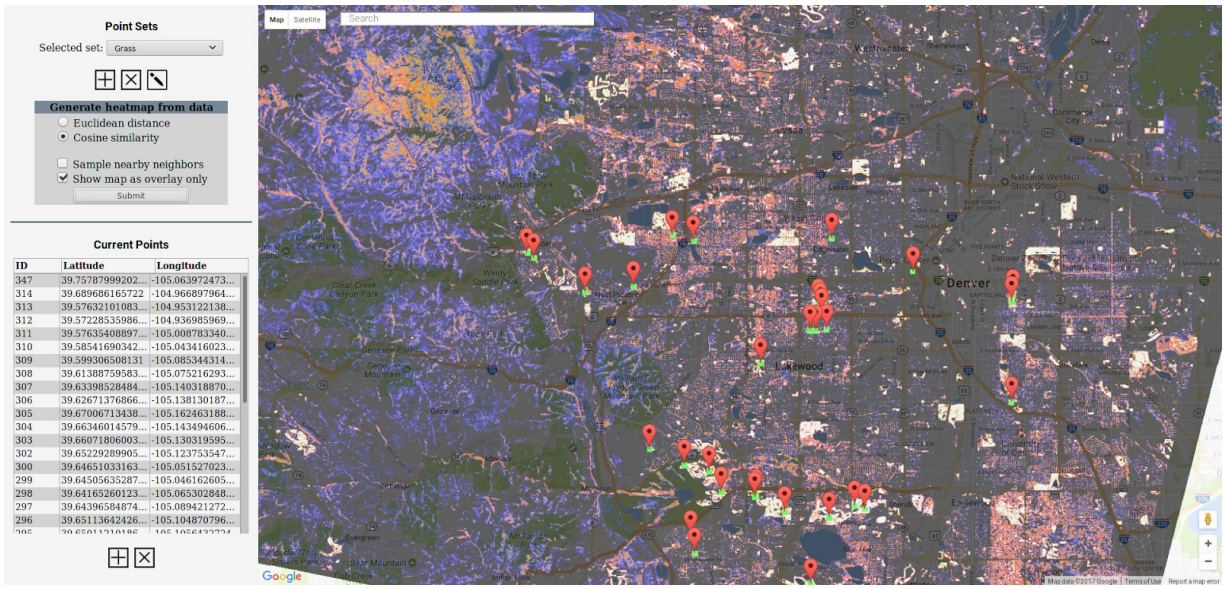

Figure 05: Screenshot of the web application with a heatmap overlay

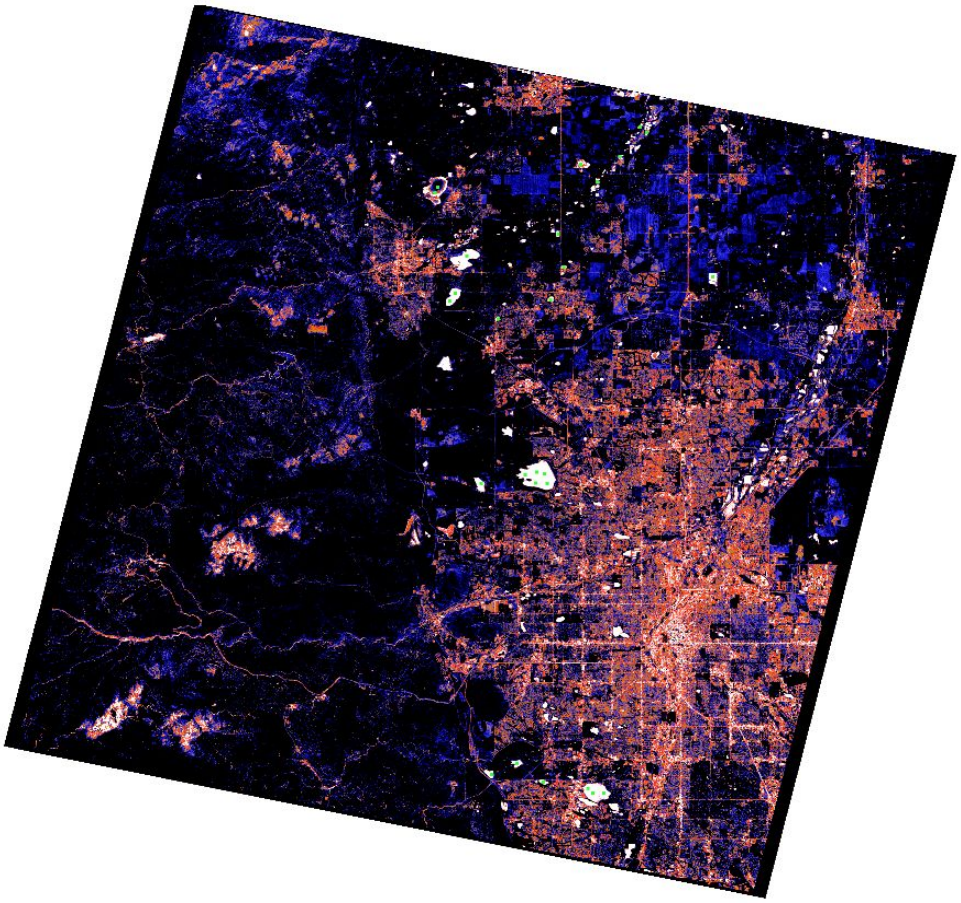

Figure 06: Screenshot of the generated heatmap showing the predicted location of water using the cosine similarity algorithm