COAL: Coal and Open-pit surface mining impacts on American Lands

Taylor Alexander Brown, Heidi Ann Clayton, and Xiaomei Wang

Group 18

CS 461: Senior Capstone Fall 2016

Oregon State University

Abstract

Mining is known to cause environmental degradation, but software tools to identify mining impacts are lacking. Researchers studying this problem possess large imaging spectroscopy and environmental quality data sets as well as high-performance cloud-computing resources. This project provides a suite of algorithms using these data and resources to identify signatures of mining and correlate them with environmental impacts over time.

CONTENTS

1 INTRODUCTION

This is the design document for COAL. It outlines the design of our system as well as the research and development priorities at each step. The COAL suite of algorithms constitutes a pipeline of data processing and presentation, and is accompanied by API documentation as well as a website for end users and the general public. Specific implementation details remain to be determined based on research, but in general the program reduces a large set of hyperspectral and geographic data into comprehensible conclusions about mining and environmental impact in human and machine-friendly formats.

Each section was authored by the corresponding team member responsible for each component identified in the technology review: Sections 2.1 and 2.2 by Heidi; section 2.3 by Xiaomei; sections 2.4, 2.5, and 2.6 by Taylor; section 2.7 by Heidi; and sections 2.8 and 2.9 by Xiaomei. The research priorities and design choices for each part of the project are discussed in each section.

2 TECHNOLOGIES

2.1 Feature extraction in AVIRIS data

All minerals have their own unique spectral signature, or relationship between wavelength and reflectance. The process of feature extraction will be done via a neural network using Spectral Python. Spectral Python can be installed via Pip or directly from the GitHub repository. The only dependency is NumPy. Spectral Python works with Python 2 and is compatible with Python 3 minus the functions ”view_cube” and ”view_nd”, which will not be used in COAL.

The first step is to load the training data. Spectral Python has different methods of opening files containing hyperspectral data and COAL will be opening the training data in ENVI format. After the data is opened, it will be normalized. This is so that when it is classified later, there is consistency in the number of spectral bands and wavelength range of the trained neural network and the data to be classified. This is done with the methods belonging to the PrincipalComponents class in Spectral Python.

In order to make a neural network functional, there needs to be a training stage. The data for the training stage will include the United States Geologic Survey Digital Spectral Library. This library is available for free in different formats, however it is important to use the format that is compatible with AVIRIS. The United States Geologic Survey Digital Spectral Library contains observed spectral signatures of minerals, combinations of minerals, and other land surfaces such as snow. Other data included will be the ASTER spectral library, which functions the same as the USGS library. The training stage will be done with the methods belonging to the PerceptronClassifier class in Spectral Python. Neural networks require a large sample size to be highly accurate. These libraries contain over one thousand spectral signatures each, so training will be considered sufficient with USGS and ASTER.

After the neural network has been trained, it will need to be saved to a file for reuse later, as it is undesirable to have to retrain a neural network every time COAL is used. There will be a function in the COAL library that makes use of Pythons pickle library in order to save the trained neural network to a file.

Some important points to consider during feature extraction is consistency between training data sets. Besides normalizing, units need to be considered. For example, USGS and AVIRIS record wavelength in different units (USGS in micrometers and AVIRIS in nanometers). If this is not addressed, the classification stage could possibly yield highly inaccurate results. This warning should also be noted in the Sphinx documentation, as the COAL library is developed for the purpose of this project as well as for use by others. Time is also a factor that could come into play here, as the

nature of neural networks requires large data sets for maximum usefulness. The training stage could take a long time, as just the USGS data contains millions of lines and some users may want to input even more training data than we will be doing for the mining analysis.

2.2 Classification of AVIRIS data

Once a trained neural network is created, as described in the description of the feature extraction phase, classification can be performed on hyperspectral data with unknown mineral concentrations. This will create the basis for later stages of the analysis of AVIRIS data for the effects of mining by identifying the minerals and land surfaces in an image which will then be analyzed for any patterns associated with mining. This be done with Spectral Python. Information on Spectral Python can be found in section 2.1.

The first step for classification is to load the trained neural network from a file. This is done with Pythons pickle library. Next comes the images to be classified. For each image to be used in classification, the file name of the image and the file name of the newly classified image must be provided as parameters. The library will take the trained neural network and the file names and iterate through each pixel in each image in order to complete the classification.

The AVIRIS image will be iterated over with the ImageIterator class in Spectral Python. An M x N array will be allocated with NumPy in order to store the classified data of each pixel. Each pixel will be normalized as per the feature extraction phase. The same function should be used for normalizing the training data and normalizing pixels. The normalized pixel will then be used as input into the neural network to be classified as a particular type of mineral or land surface identified in the training data. This is done with the methods of the PerceptronClassifier class. The classified pixel is then stored in the M x N array mentioned above. Once all pixels have been classified, the array of classified pixels will be written to a file using the save_classifier method of the envi class.

Something to consider when using this classifier for analysis is spectral separability. Some types of minerals and other land surfaces have very similar relationships between reflectance and wavelength and can be cause the neural networks accuracy to suffer if they are separated into different classes [1]. It is up to the developers to determine if the particular minerals or other land surfaces that are not very separable should be put into the same class or kept separate depending on if the minerals or other land surfaces in question are important enough by themselves in some way to warrant a slightly lower accuracy.

As with the feature extraction phase, attention needs to be paid to the units in the images to be classified and the timing. None of the Spectral Python functions will pay attention to any discrepancies between units, so it is imperative that it is addressed before classification begins. Hyperspectral images tend to be very large files that are multiple gigabytes in size and it is possible that using a long list of images as input will result in a wait time that is several minutes per image. It is important that this is considered before use.

2.3 Identify Mining

After we get the mineral identification and classification data, we need to identify the regions of mining activity. The data is further processed and possibly combined with geographic information about geology and mines. The goal of identifying mining is to accurately locate regions in which mining is taking place in a form that is suitable for both human inspection and further processing. This step creates input data for further processing of the project and the data evaluation.

To start our process of identifying mining, we need the output from the mineral identification step. The previous step has already located the regions that are containing mineral associated with mining. To distinguish mines from preexisting geology is a challenge; only rely on the existing geography data is not enough for the identification.

There are three methods mentioned after several group discussion sessions for identifying mining: mineral classification, using patterns of mineral deposits; GIS comparison, using geographic information such as mining permit boundaries or geologic maps; and a combination of mineral classification and GIS comparison, using both sources of data. To locate regions with mineral deposits that indicate mining, we have decided that we are going to use the mineral classification methods. Then, we would move to a combination of mineral classification and GIS comparison methods once we are able to complete the first step without much trouble.

The mineral classification method can be very simple, mapping the presence of a mineral may be sufficient to locate mines: minerals with low natural concentrations can be used to identify mining activity. In this case, the result is expected to be straightforward, and the time spending on the identification is short. However, the outcomes are not always favorable under this method. Still, at the very beginning, we expect to rely on this to produce some useful outputs for further application. The combination of the mineral classification and GIS comparison can be costly; it would require the most research and time. Although the outcome can be most complete, so that is why we use mineral classification method at the beginning; simply mapping the presence of a mineral may be sufficient to locate mines due to minerals with low natural concentrations can be used to identify mining activity. GIS comparison can provide more accurate results.

2.4 Preprocessing Data to Correlate Mining with Environmental Impact

After mines have been identified, we are interested in analyzing their impact on the environment. Using environmental quality GIS data from agencies such as the Environmental Protection Agency (EPA), United States Geological Survey (USGS), United States Bureau of Reclamation (USBR), and United States Department of the Interior (DOI), we would like to enable research into the environmental, health, and social impacts of mining on watersheds. This stage depends on classified mining data from the preceding stage.

The principal problem is to combine the spectroscopic-derived data with geographic data. Preprocessing this data will require using the metadata from the imagery to associate pixels with geographic coordinates. It will also require loading and handling geographic data. We have found that the Geospatial Data Abstraction Library (GDAL) provides a comprehensive toolkit for processing georeferenced vector and raster data. The team is in the process of identifying a suitable data set for environmental correlation as well as geographic information on waterway boundaries such as NASA's Global Reservoir and Dam (GRanD) collection.

To implement preprocessing, we must first combine the environmental data with the waterway boundaries. This will provide a data set of environmental measurements on watersheds. We must then identify the layer or layers we are interested in analyzing. Once we have combined the waterway boundaries with the environmental data and selected the relevant layer or layers, the data must be cropped to the boundaries of the AVIRIS flightline and converted to a compatible raster format.

Some of these procedures have already been implemented by GDAL, others we are implementing by hand in our module for GIS processing. The GDAL library handles most of the basic GIS operations such as combining the environmental data with waterway boundaries and choosing the desired layer. It was necessary to create a helper function to crop a vector source data set to the dimensions and format of a raster destination data set. It was also

necessary to create a helper function that initializes an empty data set to the dimensions of an existing image. These will be called by the subsequent stage to perform the actual correlation.

Ensuring that the input and output of our program is consistent is best accomplished by automated testing. It would be possible to apply test-driven development and write tests to specify program behavior, but we may find it more natural to implement procedures and tests alternately. At the end of the project, our tests will serve as an informal specification and an assurance of correctness. Unit tests have not currently been implemented.

Preprocessing data to correlate mining with environmental impact thus depends on a sequence of steps. We have been researching candidate data sets, using the GDAL library to perform GIS operations programmatically, and implementing helper functions to prepare the geographic data with the AVIRIS imagery in the following stage. Ensuring that these functions meet our requirements will involve automated tests. The end product will be preprocessed geographic information that can be used to make correlations between mining and environmental impact.

2.5 Feature Extraction to Correlate Mining with Environmental Impact

Given preprocessed geographic information and mining-classified imagery, we would like to facilitate correlations between mining and environmental impact. For our case study we would like to identify regions where mining overlaps with water pollution. The result will be a georeferenced raster file which can be used to print maps or to generate more sophisticated analyses. Automated tests will help enforce our assumptions about these procedures, but data that is generated will have to be analyzed in person and adjusted according to utilitarian as well as aesthetic preferences to meet the needs of the research task at hand.

To derive the results, we will allow the application programmer to define a function that combines geographic data with imagery to produce an output. Our case study simply overlays the data sets. To do this, we plan on comparing each mine-classified pixel with the corresponding water pollution pixel to generate a resulting map of pixels where the two conditions overlap. This is a simplistic model and not a true correlation, but it should suffice to allow us to develop and test our library. Environmental data scientists will be able to define more sophisticated functions over the images.

This module will call upon procedures developed in the previous step as well as libraries for processing hyperspectral imagery and geographic information. We anticipate to continue using Spectral Python and GDAL to handle the data, although we may need to turn to additional libraries to facilitate the kind of analysis researchers expect in practice. The libraries we are using are relatively mature, however we have a GitHub Organization in the event that we need to patch third-party libraries. One such patch would be to add GPU acceleration to the machine-learning algorithms included with Spectral Python in order to optimize training and classification, such as by reimplementing the perceptron classifier with the OpenCV library. The GDAL library generates formats compatible with many GIS programs, so we may leave the data visualization up to the user.

The routines for processing geographic data developed in the preceding module are not clearly separated from some of the processing that occurs during the correlation stage. Currently the combine function is implemented in the GIS code and called by both the environment code and the prototype mining identification code. Thus our original intuition that the modules may be separated appears to be justified by the current evolution of the software.

The output formats may include maps and charts, but as mentioned these may be left to the user for post-processing. The GDAL library provides broad compatibility with multiple geographic information formats and thus our code could be extended to support arbitrary formats. The simplistic analysis that we are pursuing as a case study may not lend itself to the sorts of complicated diagrams that we imagined in previous iterations, however environmental data scientists

who make use of our code could post-process the output for more sophisticated analyses. In general we expect the output of this module to be a raster file corresponding to the AVIRIS flightline in question.

Automated tests will help ensure the correctness of parts of the program, but the end result will be best analyzed in person. Components that could be tested include those responsible for converting input formats to a predictable binary output format. Sample data can be provided and tested to ensure that results are consistent with expectations. However, the overall usefulness and quality of the data visualizations must be judged subjectively. This may require loading the resulting imagery into GIS applications or other visualization programs in order to get a sense for the shape of the data.

Deriving correlations between mining and economic impacts is the purpose of this feature extraction stage. We have documented and begun to implement a set of algorithms for combining the data sets in an extensible way. The results will be in standard formats suitable for display in GIS applications or for post-processing. Automated tests will aid development, but the quality of the results will also depend on subjective determinations.

2.6 Rank and Document Changes Over Time

Ranking and documenting changes over time is the final step in the data processing pipeline. After identifying minerals, mines, and associated environmental impacts, further analysis is possible by providing a historical context to environmental changes. Implementation of this step could be as simple as instructing the end user to run the program several times over separate data sets, however this would not provide effective usability. A more sophisticated implementation would take as input a historical data set and generate trends automatically. The user will be expected to select a set of images and environmental data sets to examine, and our procedure would automate the analysis. As noted in the preceding section, geographic information formats with broad compatibility allow the user to post-process data as desired.

Before development on this step can begin, the preceding stages of data processing must be complete. The design choices and implementation details made at each step will affect the options available for temporal analysis. Data formats and representations will influence the kinds of results that can be produced. Program structure will determine how reusable certain software components are, a consideration that will be important when combining them to analyze changes over time.

The simplest interpretation of documenting changes over time is to process historical data and generate representations for each time period separately. A note in the user documentation would describe how to execute the program over multiple data sets. As mentioned, the user would be expected to select images and data which are temporally related. The user would have full power to customize the processing step, but for simplicity and usability we may like to implement a means of automating this processing.

More sophisticated methods could be implemented to relieve the burden of repetitive tasks at the cost of additional research and development time. For example, this module could receive arguments or configuration files containing multiple data files and run the analyses automatically. It could even be configured to process new data as it is received from the instruments on an ongoing basis. Designing the program with automation in mind would allow new options for output such as animations and generating temporal graphs which would not be possible if correlations are made in isolation. We expect the output format to be sets of georeferenced raster files containing the results. Post-processing these files could be implemented, for example to generate animations, however it may be more natural to leave this up to the environmental data scientists whose research applications we may not be able to anticipate.

We now expect that this module will simply call upon the previous modules in the pipeline in order to provide automation. Additional third-party components could be used if we implement more sophisticated output processing. However, compatibility with existing GIS applications may satisfy the needs of the end user.

The research and development of this module will produce results which cap off the suite of data processing algorithms provided by COAL. Ranking and documenting environmental changes over time combines the information from all earlier steps into a unified whole for producing temporal analyses to inform environmental data scientists and other interested parties. Design and implementation could be simple, but more sophisticated approaches could be devised to enhance usability with automation and post-processing.

2.7 API Documentation

The Sphinx API documentation will be developed throughout the entirety of the project, with each team member documenting their code contributions as they develop. The code written will include docstrings for all functions that will include, at minimum, a short description of the purpose of the function, description of each parameter, and the input and output of the function. Further docstrings in the code will include explanations of algorithms and structures. We will then use reStructuredText (.rst) files to integrate any function descriptions with further elaboration and any examples, figures, and/or images. We will have a main page for the documentation that includes a table of contents, a search bar, and a general description of the API. We will also have a page regarding how to use COAL's API and a page for each library. The goal of the documentation is to be useful to a person who wants to use it in their software project but does not necessarily want to know everything that is going on in explicit detail. It is a description of the API as a black box for the most part, though the source code is available to those who want more detail and is linked on the homepage 2.9.

The main page has an auto-generated table of contents and search bar, so the only thing we will be writing on the main page is the general description of the API. The page regarding using the COAL API will include information on dependencies, compatibility with AVIRIS imagery, and examples showing how to get started with COAL. Each page documenting a library will have a short description of the library at the top, followed by a description of each function (documented in the code itself with docstrings). If applicable, any concerns regarding compatibility with other libraries or performance should be addressed here. This is particularly important since image processing takes a lot of computing power. The specifics of algorithms and statements are not necessary for the auto-generated documentation. We will put comments in the source code where necessary for people who want more detail, but the API documentation will only include specifics if they raise any kind of concern.

Testing will be relatively simple to make sure the API documentation is working correctly. Syntax errors are raised by Sphinx if they are encountered, so they are easy to check for. We will also be doing spell checking on all of our documentation to make sure it is clear and professional. We will also check to make sure that the documentation is output in a place that is accessible to users.

Members of this team have used Sphinx documentation before and found it elegant, flexible, and easy to read and write. It is also the method that Python uses for its documentation, which is frequented by members of this team and is seen as nicely formatted and organized. Sphinx is also compatible with Python 2 and Python 3, which makes it a good match for us if we ever decide we want to switch completely to Python 3.

2.8 Static site generator

We will use a static site generator to provide a fast and simple back-end for COAL's homepage. GitHub let users create one site per GitHub account or organization, which is convenient for us due to that we have to keep the page's contents dynamically.

What we will do first is create a GitHub page. The GitHub page can host personal, organization, or project pages directly from a GitHub repository. GitHub pages use github.io domains which are served over HTTPS. The master branch of the repository is responsible for the publishing; this can be selected from the GitHub Pages site repository under the settings option. Once we have it linked, we will start to build our website. The advantage of the static site generator is that it does not need to deal with databases. The static site generator mostly works with JavaScript and HTML and that is why it has fast speed and relatively better performance. The main purpose of our website is to display the resources we need and make it convenient for our team to perform our tasks, so a static site generator is the perfect choice for us. Our site will not require many complicated functionalities, so a more complex back-end is not needed. Also, because we need to change the content on the home page often as the project continues, static site generators tend to have excellent version control for the content.

As the technology review states, we are going to use Jekyll as our static site generator. Jekyll is a simple, blog-aware, static site generator perfect for personal, project, or organization sites. We chose Jekyll as our static site generator because it has high compatibility with GitHub pages, which makes the work easier to do. To start to use Jekyll, we could simply install it using the command line provided by the official website. Once we get the resources we need, we will create the back end of our home page. We do not have any experience with using Jekyll, but the guide on the Jekyll website is simple to follow. If we do not need to run Jekyll on the local computer, we could only build the directory structure. The existing of the GitHub page makes our job easier. Right now, we only use Jekyll to organize our directory and make our blog post function easier to implement.

Overall, using a static site generator is not a difficult task and there are a lot of resources online. Choosing a static site generator will save us time with maintenance and server resources.

2.9 Front-end framework

Due to the continuous nature of our project, we will need a website to show our work and progress both throughout the school year and possibly beyond. There will not be much content to put on the site toward the beginning of research and development, but the content will be filled out as the project progresses. The website serves the purpose of showing our work and presenting our project in a clear and succinct way.

We use plain HTML to build the skeleton of the website, then we put the necessary information on, such as links to our GitHub pages, links to our project description that is located on our course website, the pages to put all of our papers, and links to our developers' GitHub page. All of those links are available on the navigation bar, so it is convenient for us to access them. We will be using a template for a navigation bar that she has used before so that sh can build a simple page before filling everything inside. Also, we put an introduction video on the front page.

Then, based on what is required, or what we want to put on our website later, we may need to alter the content. For the styling, we will use CSS. Our team has discussed that we would mostly use Bootstrap for our front-end framework, for the abundance of its themes and functionalities that we might need later in the project development.

Bootstrap is easy to use and works well with HTML, CSS, and JS; we choose it because of it saves us time so we can build a basic page very quickly. We also need some pictures related to mining or AVIRIS to make our site more

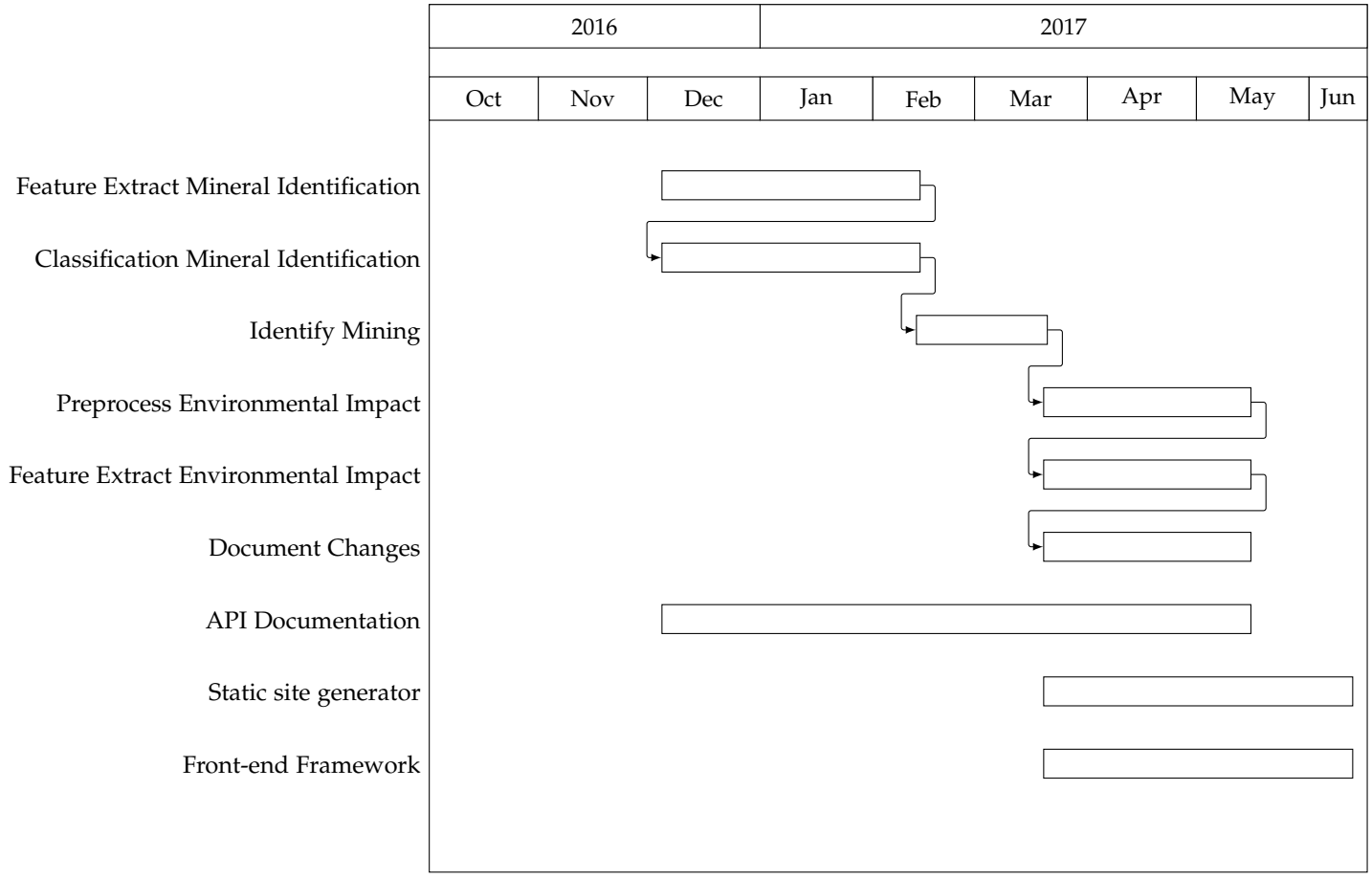

Fig. 1: Gantt Chart: Timeline of Project Tasks

aesthetically pleasing and to provide interest. Taylor has provided one earlier. Due to the reason that most of the images are too colorful to use as a background page for a website, I decide to make it more transparent so it can fit the other elements of the site.

As the project continues, we will update our homepage on a weekly basis as well as the link to our individual weekly updates for required for the computer science senior design course.

3 TIMELINE

The Gantt chart pictured in Figure 1 describes the project timeline. Dependencies among each stage of the project are depicted by arrows and approximate implementation dates are indicated by the date range. Some components may be able to be developed independently while others may require a sequential approach. In general, however, our development will follow the data from initial processing to final representations. API documentation will parallel the development of our source code using comments to automatically generate browsable reference material. The website will provide a hub for developers, end users, and the general public describing our project, and its development will be relatively independent of the algorithmic components.

4 Conclusion

This document has outlined the design of the COAL project and the suite of algorithms and documentation it provides. The major components of the system were broken down and assigned to each team member who described in each section the research priorities and design decisions involved. Processing large datasets of imagery and geographic information require a significant investment to study and develop solutions. However, the results will provide a wealth of information for environmental data scientists and other concerned parties looking for associations between mining and environmental damage. Further revisions of this document will reflect specific choices made as the project matures.

5 Glossary

AVIRIS	Airborne Visible/Infrared Imaging Spectrometer
COAL	Coal and Open-pit surface mining impacts on American Lands
docstring	Documentation string. In Python, they are comments that start and end with three quotation marks.
GIS	Geographic Information System

6 Revisions

6.1 Feature extraction in AVIRIS data

Revisions to Section 2.1: Scrapped blob detection and wrote about training neural networks.

6.2 Classification of AVIRIS data

Revisions to Section 2.2: Rewrote most of the section and made it more specific to the development at this time.

6.3 Identify Mining

Revisions to Section 2.3: No revisions.

6.4 Preprocessing Data to Correlate Mining with Environmental Impact

Revisions to Section 2.4: Completely revised to describe our new knowledge about geographic information data.

6.5 Feature Extraction to Correlate Mining with Environmental Impact

Revisions to Section 2.5: Completely revised to describe our environmental case study and the current implementation of the code.

6.6 Rank and Document Changes Over Time

Revisions to Section 2.6: Completely revised to present our better understanding of the batch- and post-processing.

6.7 API Documentation

Revisions to Section 2.7: No revisions.

6.8 Static site generator

Revisions to Section 2.8: To clarify what kind of functions we need Jekyll to provide, and to how to get started to use it.

6.9 Front-end framework

Revisions to Section 2.9: Add details about what is on the navigation bar. Add the details of what changes of the website.

REFERENCES

[1] W. Amidon. (2014, March 17). *v26 evaluating spectral separability in ENVI* [YouTube video]. Available: https://www.youtube.com/watch?v=y685Km9njys

[2] W. Amidon. (2014, March 17). *v25 introduction to classification in ArcMap and ENVI* [YouTube video]. Available: https://www.youtube.com/watch?v=OBJKGWocM6Q