482 PROJECT PROPOSAL

Giovany Addun

Conner McCloney

Ryley Rodriguez

INTRODUCTION

In natural resource management, land cover and land use are terms used to describe what types of natural and man-made features compose an area of land. This information is typically presented as land cover/land use (LCLU) maps, which are typically created using machine learning techniques on remotely sensed data. These maps are important to people like city when planning city growth, or ecologists when tracking invasive species, or agronomists when deciding what crops to plant or soil treatments to apply.

Since 1972, the Landsat program has launched 8 observational satellites. These satellites continuously orbit the earth, collecting ground reflectance values for different wavelengths of light. Because the Landsat program has been continuously making observations for so long, it contains a trove of data well suited for making LCLU maps out of.

Although there exist tools for converting Landsat data to LCLU maps, most of these tools are quite expensive and many of the free tools have strong limitations. This makes it difficult for citizen scientists to conduct studies using LCLU data or remotely sensed data in general.

Given this difficulty, we have chosen to implement several LCLU classification techniques using only free and open-source software. We have chosen 5 classification techniques to implement and compared their relative strengths in terms of precision, recall and accuracy while making LCLU classifications on 3 different geographic location.

QUALIFICATIONS

RESUMES

[See appendix A]

PROJECT BACKGROUND

LANDSAT

Landsat is a joint NASA, USGS project to collect spectral reflectance data of the Earth with earth orbiting satellites. There have been 8 satellites launched under this project since 1972 [1], making Landsat data the longest continuous record of satellite observations. Each satellite has an array of sensors that each collect reflectance values for a specific range of wavelengths of light. Often, these sensors and the wavelengths of light that they capture are referred to as “bands”. These sensors have a spatial resolution of 30mx30m per pixel, with an image consisting of a grid of roughly 8000 x 8000 pixels per band.

Landsat data are distributed as .Tiff or .GeoTiff file formats [2]. These data a made available for free to the public through the USGS. Additionally, both Google and Amazon host the entirety of Landsat public images. We have chosen to use the google public dataset [3] for this project, because there is a convenient python API (Application Program Interface) which allows the process of data collection to be automated.

The pixel values of these data are stored in 7-bit or 8-bit unsigned integers as DN (Digital Number) values [4] [5], integers ranging from 0-255. These DN values are completely relative to the sensor and band from which they are captured. This means that a DN value of 5 (for example) for one sensor on one band on a Landsat 7 image could correspond to an entirely different surface reflectance for a DN value of 5 on a Landsat 8 image on the same band. Therefore, before meaningful analysis can be conducted on these data, they must be converted from DN’s to absolute values [6]. For our study we have chosen to use TOA (Top of Atmosphere) reflectance values. Using TOA reflectance will allow us to compare images from different times, locations and captured by different satellites.

[See Appendix B.4 for example Landsat image]

CLASSIFICATION

Classification is the task of assigning class labels to objects which are to be classified. In the context of machine learning, classification methods can be divided into 3 categories: Supervised, Unsupervised, and Semi-supervised [10]. In each case, a machine learning model must be provided some set of data to train the model to make accurate classifications. In the context of our project, we plan to classify individual pixels of Landsat images (representing a 30m x 30m area on the surface of the Earth) into one of three classes; Developed, Undeveloped, and Water.

In supervised models, a set of pre-labeled training data are given to the classification model. The model is trained by making predictions about a sample's class label, checking the sample's actual class label, and then making corrections to the model if the prediction was incorrect.

The user of a machine learning model for classification must decide what amount of training data is used and how many training iterations to use to train the model. Additionally, the user must devise a way to validate their model, that is, to measure the accuracy of its predictions prior to applying the model. These decisions depend largely on the type of data, model deployed, and specific application of the machine learning model.

For our project we generated training datasets of approximately 200,000 data points per geographic location. We used 4-fold cross validation on each of 30 independent trials per (algorithm, location) pair.

MULTILAYER PERCEPTRON

Multilayer perceptrons are a type of feed-forward neural network. For our implementation we chose to use gradient descent as the method of backpropagation and hyperbolic tangent as the sigmoid function. We chose to compare multilayer perceptrons because they perform generally well among with image classifiers.

RANDOM FORESTS

Random Forest is a supervised learning technique that can be applied to regression and classification [11]. A random forest model consists of a large number of randomly generated decision tree classifiers. During classification, each decision tree within the model is presented with the object to classify, and then reports the expected class label for the object. The Random Forest model then tallies the reported class labels from each decision tree and makes a prediction of the true class of the object based on these reports. In most implementations of Random Forest, each tree accounts for a single unweighted vote for a class, and the Random Forest model selects the predicted class label with the most votes.

We chose to use Random Forest as our classification model because it has been demonstrated to be consistently well performing over a wide variety terrain types within the context of Landsat image classification [12]. Additionally, Random forest has a built-in validation feature, Out-of-Bag error estimation, that many other models do not, making it preferable for our application.

CATBOOST

Catboost is an algorithm built by the search Engine Yandex. Like RandomForest, it is also an ensemble of voting decision tree classifiers. However, unlike RandomForest, Catboost uses gradient boosting to iteratively train the trees to minimize the error to an arbitrary loss function. For our implementation we used accuracy as the loss function and 200 training iterations. We chose to compare the Catboost classifier because it is a fairly new algorithm and has been shown to be versatile in its applications.

NAÏVE BAYES

Naïve Bayes Classifiers use a training dataset and Maximum A Posteriori estimation to estimate the probabilities of classes and sets of features, and then uses Bayes' theorem to predict a class given a set of features. Naive Bayes models are "naïve" because they assume conditional independence of pairs of features and classes. Naïve Bayes models have been shown to be particularly well suited for Natural Language processing tasks such as sentiment analysis.

K-NEAREST NEIGHBORS CLASSIFIERS

K-nearest neighbors classifiers maps the sets of features in the sample data to vectors in some feature space. When a constructed model classifies a datapoint, it maps the datapoint to the feature space and examines some number, k, of it's nearest neighboring points and the class containing most of the nearest neighbors is assigned as the class label. For our implementation we chose to use k of size 30 and L@Norm as the distance metric

WORK SCHEDULE

We have broken our project into 4 phases; Data collection (1), Preprocessing (2), Modelling (3), and Interpreting Results (4).

Once we have moved to the third phase, we will meet in person weekly to discuss progress and ensure we are meeting deadlines. During phase 3 we will decide specific responsibilities based on schedule, availability, and need.

We have included a visual timeline in this proposal

[See appendix B.1]

PHASE 1 & 2

During the first two phases, Data collection and Preprocessing, we will be focused on collecting data. This will data collection will consist of

- Researching suitable sites with significant urban growth
- Compiling a list of suitable images (Landsat product IDs) by visually searching images on USGS EarthExplorer
- Collecting Landsat images from the Google public Landsat dataset
- Transforming Landsat images to numpy arrays of TOA values
- Using Google earth to collect ground through truth training data
- Compiling a database of training data, latitude/longitude coordinates, and class labels
- Building and training a classification model
- Applying the classification model to Landsat images to create a LUCC maps for each image.
- Collecting demographic data for the chosen sites where available (population growth rate, average income, etc…)

For these phases we will conduct weekly check ins by email.

PHASE 3

During this phase we will research classification models, and apply those models to collected Landsat images. A majority of the software development for this project will occur during this phase. This being the case, we plan to meet in person at least once a week to discuss progress on the project. Additionally, we will be in constant communication through collaborative editing tools such as Git and Google Docs.

PHASE 4

During this phase we will conduct significance tests on the results of our predictive models. We will then compile our findings into a written report, to be accompanied by a poster and 30 minute Powerpoint presentation

As in Phase 3, we will meet at least once weekly in person to discuss our progress in addition to being constantly in communication through collaborative editing tools.

INDIVIDUAL RESPONSIBILITIES

Most of the responsibilities for this project will be divided evenly between all team members. Below is a list of shared responsibilities.

- Each team member is responsible for collecting Landsat images and ground truth data for one urban site.

- Each team member will contribute in researching classification methods
- Each team member is responsible for contributing to reports and presentations.

GOIVANY ADDUN

Giovany will also be responsible for

- Developing the data collection software package
- Developing Classifier implementation

RYLEY RODRIGUEZ

Ryley will also be responsible for

- Developing classifier implementation

CONNER MCCLONEY

Conner will also be responsible for

- Choosing and implementing statistical tests of significance for results

PROPOSAL STATEMENT

For our project we propose to compare the performance of 5 machine learning techniques in terms of precision, recall, and accuracy. To accomplish this, we performed 30 independent classifications of our generated datasets per (algorithm, location)-pair and recorded the respective metrics. Additionally, we used 4-fold cross validation.

FUNCTIONAL REQUIREMENTS

We have identified the following as functional requirements for our applications

- Able to predict land use over time
- Able to read tiff files
- Runs on Google Cloud
- Uses machine learning models for prediction and classification
- Compiles Landsat images into numpy arrays

NON-FUNCTIONAL REQUIREMENTS

We have identified the following non-functional requirements of our project

- Time efficient
- Easy to use
- Well documented (Code)
- Organized
- High quality (Code)

PERFORMANCE REQUIREMENTS

PLATFORM

We intend to run all of our computations on the Google cloud compute platform.

On this virtual machine platform, we intend to perform our analysis in the Python data science environment, Anaconda.

SCALABILITY

We have no intention of extending our project beyond the scope outlined within this proposal. Therefore scalability is not a design consideration for this project.

RESPONSE TIME

- The Python program to collect data given a text file should process a given line and generate a URL in less than 0.5 seconds
- The Python preprocessing program constructs a numpy array representation of an image in less than 2 minutes per image

INTERFACE REQUIREMENTS

- Python script will generate URLs from imaged IDs in text file
- Python script will request and grab images from Google's public Landsat dataset
- VM Instance will upload files to project cloud storage

ARCHITECTURAL DESIGN DOCUMENTS

We have chosen to use the Google cloud platform as our development environment. Specifically, we are using virtual machines on the cloud compute engine and cloud storage buckets for our data.

[see appendix B.2]

DEVELOPMENT STANDARDS

METHODOLOGY

DESIGN PATTERNS USED

We have chosen to use the Abstract Factory [15] design pattern for our data preprocessing application. We chose this design pattern because it addresses the challenges of working with Landsat data from a range of satellites. Each data set has some routines that can be applied to any data from any satellite. However, the process of calculating TOA (Top of Atmosphere) reflectance from DN's (Digital numbers) changes depending on the specific satellite used in the collection of the data.

[see appendix B.3]

RELEVANT UML DIAGRAMS

[See appendix B.5]

SOFTWARE DEVELOPMENT LIFECYCLE

We have chosen to implement the Big Bang model [16]. We chose this software development lifecycle model for several reasons.

The most significant factor in the decision to use this model is that it is a simple model to implement. At the highest level of abstraction, the software of our project only has 4 functional requirements: Collect and Store Data, Preprocess Data, Classify Data, and Interpret results. In addition to this, our project will not be subject to changes in customer requirements because the development team is essentially its own customer. Furthermore, the software that will be developed for this project is intended to serve a single, one-time-use purpose and therefore it is not necessary to take in to considerations factors such as maintainability of the software during the development process. The combination of simplicity in design and constancy of requirements makes using the Big Bang model for our project relatively low risk.

METHOD OF ANALYSIS

We initially intended to use two-way ANOVA to analyze our results. However, upon examining the residual plots we saw strong evidence that homogeneity of variance and normality assumptions were violated and therefore ANOVA was not appropriate approach. We additionally conducted Levene's test and Shapiro's test to confirm these violations.

There were 3 cities: Austin, Mesa, and Palm Coast, and five algorithms: Bayes, Catboost, Neighbors, Neural Networks, and Random Forest, resulting in 15 total combinations. For each combination, a sample size of n = 30 observations was generated, and the mean value for each of the three metrics of interest was recorded. To test the differences between all 15 samples, we will first outline our hypotheses tests. Our null hypothesis states that for any combination, the true mean of the metric of interest for each are equivalent, or H0 : $\mu_{i,j} = \mu_{k,l}$ where i, j are a given City, and k, l are a given algorithm where i ≠ j or k ≠ l. Whereas our alternative hypothesis states that they are actually different, or HA : $\mu i_{,j} \neq \mu_{k,l}$, given the same constraints.

Because of the nature of the data, we chose to use pairwise comparisons of trials using bootstrapping.

CONCLUSIONS

Most of our results for all three metrics are statistically significant, or we have sufficient evidence to reject H0 for the vast majority. In total we had (3 metrics) x (105 pairwise comparisons) = 415 comparisons, of which only 8 were not statistically significant, with p-values > 0.05.

Plots for performance of (algorithm, location)-pairs for each metric in [appendix B.6]

Statistical tests conducted in [appendix C.2]

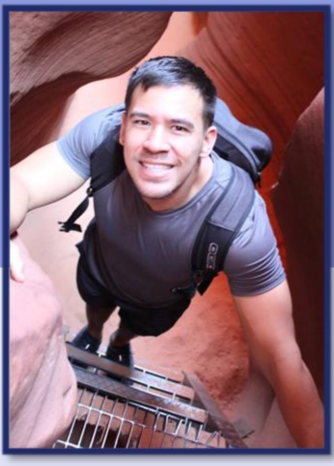
REFERENCES

1. Landsat Missions Timeline. (n.d.). Retrieved from https://landsat.usgs.gov/landsat-missions-timeline
2. Landsat Level-1 Data Products. (n.d.). Retrieved from https://landsat.usgs.gov/landsat-level-1-standard-data-products
3. Landsat Data | Cloud Storage | Google Cloud. (n.d.). Retrieved from https://cloud.google.com/storage/docs/public-datasets/landsat
4. Landsat 8 Data Users Handbook. (n.d.). Retrieved from https://landsat.usgs.gov/landsat-8-data-users-handbook
5. Landsat 7 Data Users Handbook. (n.d.). Retrieved from https://landsat.usgs.gov/landsat-7-data-users-handbook
6. Young, N. E., Anderson, R. S., Chignell, S. M., Vorster, A. G., Lawrence, R., & Evangelista, P. H. (2017). A survival guide to Landsat preprocessing. *Ecology,98*(4), 920-932. doi:10.1002/ecy.1730
7. Lambin, Eric F, & Geist, Helmut. (2006). Modeling Land-Use and Land-Cover Change. In Land-Use and Land-Cover Change: Local Processes and Global Impacts (Global Change - The IGBP Series, pp. 117-135). Berlin, Heidelberg: Springer Berlin Heidelberg.
8. Ozturk, D. (2015). Urban Growth Simulation of Atakum (Samsun, Turkey) Using Cellular Automata-Markov Chain and Multi-Layer Perceptron-Markov Chain Models. Remote Sensing, 7(5), 5918-5950.
9. Chen, Chi-Farn, Son, Nguyen-Thanh, Chang, Ni-Bin, Chen, Cheng-Ru, Chang, Li-Yu, Valdez, Miguel, . . . Aceituno, Jorge. (2013). Multi-Decadal Mangrove Forest Change Detection and Prediction in Honduras, Central America, with Landsat Imagery and a Markov Chain Model. Remote Sensing, 5(12), 6408-6426.
10. Mohammed, M., Khan, M., & Bashier, E. (2016). Machine Learning (1st ed.). CRC Press.
11. Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32.
12. Lawrence, & Moran. (2015). The AmericaView classification methods accuracy comparison project: A rigorous approach for model selection. Remote Sensing of Environment, 170(C), 115-120.
13. Dorigo, M., & Stützle, T. (2004). Ant colony optimization. Cambridge, Mass.: MIT Press.
14. Agapie, A., Andreica, A., & Giuclea, M. (2014). Probabilistic Cellular Automata. Journal of Computational Biology, 21(9), 699-708.
15. Gamma, E. (1995). *Design patterns : Elements of reusable object-oriented software* (Addison-Wesley professional computing series). Reading, Mass.: Addison-Wesley. Kazim, A. (2017).

16. Kazim, A. (2017). *A Study of Software Development Life Cycle Process Models*. International Journal of Advanced Research in Computer Science, 8(1), International Journal of Advanced Research in Computer Science, Jan 2017, Vol.8(1). Unpingco, J. (2016).
17. Machine Learning in *Python for probability, statistics, and machine learning*.
Switzerland: Springer International Publishing.

APPENDIX A (INDIVIDUAL QUALIFICATIONS)

Giovany Addun



Experience

07/2014 – 07/2016
Service Representative • Social Security Administration

Assisted customers from a wide range of backgrounds in understanding the laws and rules regarding social security benefits

10/2012–04/2014
Scout sniper • U.S. Army

Conducted reconnaissance

04/2009–10/2012
Radio Telephone Operator • U.S. Army

Maintained communication and relayed information between dozens of groups of soldiers during tactical operations

Education

Montana State University, Bozeman Mt
B.S. Computer Science (minor in Mathematics)
(Expected completion 05/2019)

- GPA 3.37 on a 4.00 point scale. Coursework includes Databases, Machine Learning, Probability theory

Leadership

- Army Warrior Leaders course
- Army Cold Weather Leaders Course

Relevant skills

- Experience with Google Cloud Platform
- Familiarity with many Python Data Science packages and their domains of applications
- Excellent communicator and considerate listener

Contact

877
Bozeman, Mt 59715

907-227-2903

Giovany.m.addun@gmail.com

Programming Languages

- Python
- Android Studio
- Java
- Ruby
- R

References

[Available upon request.]

Giovany Addun

Contact

877
Bozeman, Mt 59715

907-227-2903

Giovany.m.addun@gmail.com

Conner McCloney

(406) 539-1214 • cmccloney@hotmail.com • linkedin.com/in/conner-mccloney

EDUCATION

B.S. in Computer Science expected December 2019
B.S. in Mathematics-Statistics
Montana State University

Cumulative 3.80 GPA
Member of Upsilon Pi Epsilon, Computer Science Honor Society
Member of Pi Mu Epsilon, Mathematics Honor Society
Member of Dean's List four years consecutively
Outstanding Scholar Award recipient

PROJECTS

Remote Control Car

•The project code can be accessed at: https://github.com/cmccloney/RC-Car
•Programmed an Adafruit IMU board to wirelessly operate an RC Car using keystrokes from, and transmit statistics back to, a computer
•Developed a GUI that would display real-time measurements and a timer, along with a second screen that would display averages, maximums, and minimums
•Worked in a four-person team with members from other engineering disciplines to successfully complete the project
•The IMU board was programmed in Arduino, while the GUI was developed using Java

Football Data Analysis

•The project code can be accessed at: https://github.com/cmccloney/Python-Football-Data-Analysis
•Using the requests and BeautifulSoup libraries in Python to scrape HTML content from webpages
•Using the pandas library to manipulate and display data on player salaries and ages
•Display a selected team's recent and upcoming games along with team-specific statistics in a multi-page Tkinter GUI
•Currently a work in progress, plan on analyzing and customizably comparing statistics on all professional football teams in a Dash framework

Monopoly Board Game

•The project code can be accessed at: https://github.com/cmccloney/Monopoly-Project
•Wrote a fully functioning software version of the Monopoly board game in Java
•Created a GUI framework for the player(s) to interact with, including the option to customize the board and street names used
•Used UML Diagrams to organize all necessary classes and functions, and implemented JUnit testing on classes and functions to ensure functionality of the program

ACHIEVEMENTS AND SKILLS

Certificates	CompTIA A+, CompTIA Network+, CompTIA Security+
Programming Languages	Java, C/C++, R, Python, HTML/CSS
Software Tools	Git/GitHub, Eclipse, NetBeans, RStudio

RYLEY M RODRIGUEZ

2962 Oliver St | Bozeman, MT | (719) 371-4957 | ryleymichael127@live.com

EDUCATION

Montana State University

Major: Computer Science	Minor: Mathematics
Expected Graduation: Spring 2019	Cumulative GPA: 3.77

Fort Lewis College

Major: Mathematics	Minor: N/A
Transfer Date: Spring 2016	Cumulative GPA: 3.63

RELATED COURSES

Database Systems**,** Artificial Intelligence, Software Engineering, System Administration

SKILLS

Languages

Java / C / C++ / Prolog / Ruby

Developer Tools

GitHub / IntelliJ / Eclipse / NetBeans

CLUBS & HONORS

Upsilon Pi Epsilon Honor Society Member | MSU	**Spring 2018**
College's Dean List | Fort Lewis College	**Spring 2015**

WORK HISTORY

Office of Sustainability ~ MSU | Fall 2016 – Present
Recycling Assistant
- Collaborating with team members to sort recycling
- Maintaining a time-sensitive schedule

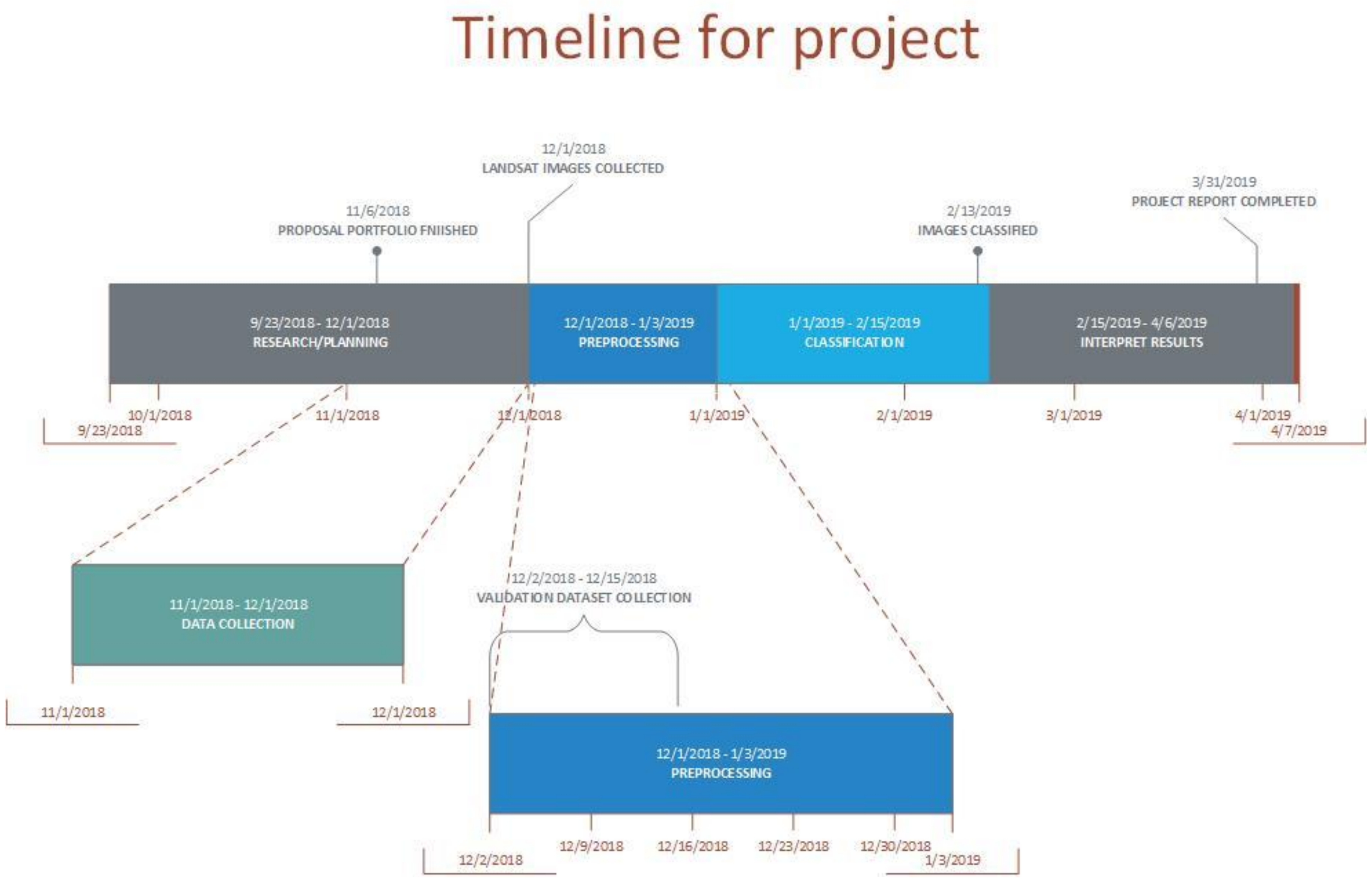
F/V Ocean Harvester ~ Alaska | Summer 2016
Deckhand
- Communicated with customers and supervisors
- Combined the strengths of crew to improvise and solve problems

Admissions Office ~ Fort Lewis College | Spring 2015 – Spring 2016
Communications Manager's Assistant
- Assembled mail packets and input student records into computer
- Enhanced attention to detail and communication skills

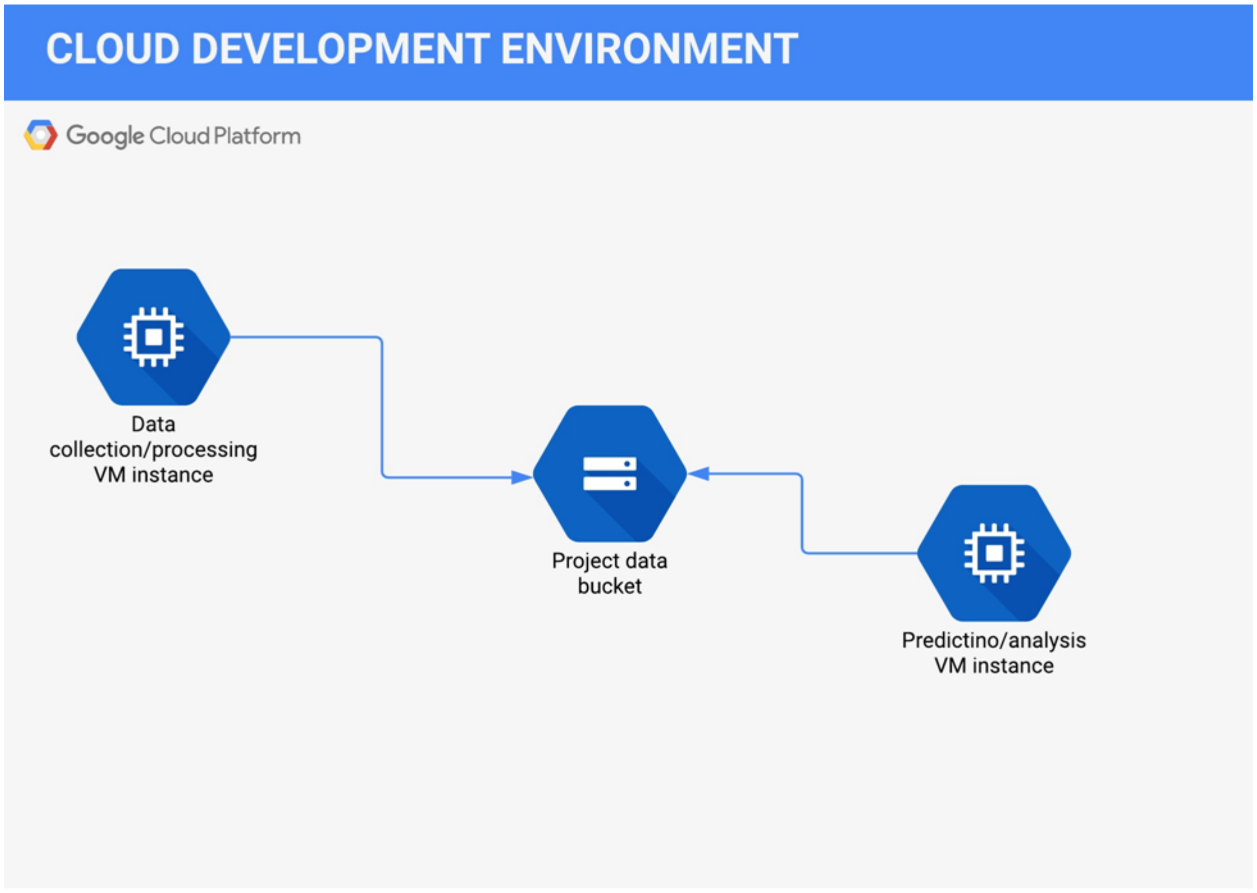
All Solar Inc. ~ Penrose, CO | Summer 2010 – Summer 2015
System Installer
- Performed physical labor and retrieved tools
- Sparked my interest to create applications to make people's lives easier

APPENDIX B (DIAGRAMS)

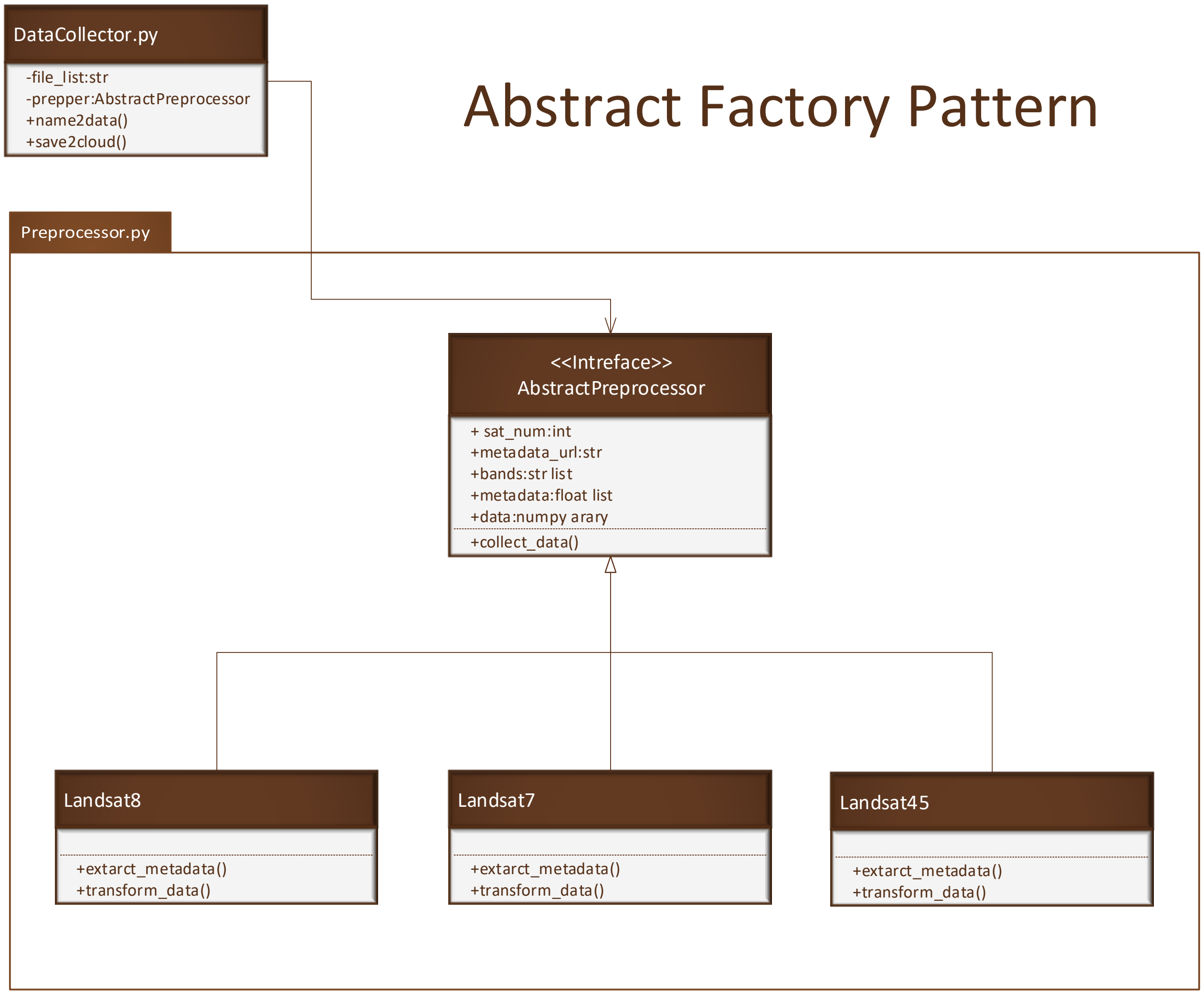
B.1 (PROJECT TIMELINE)



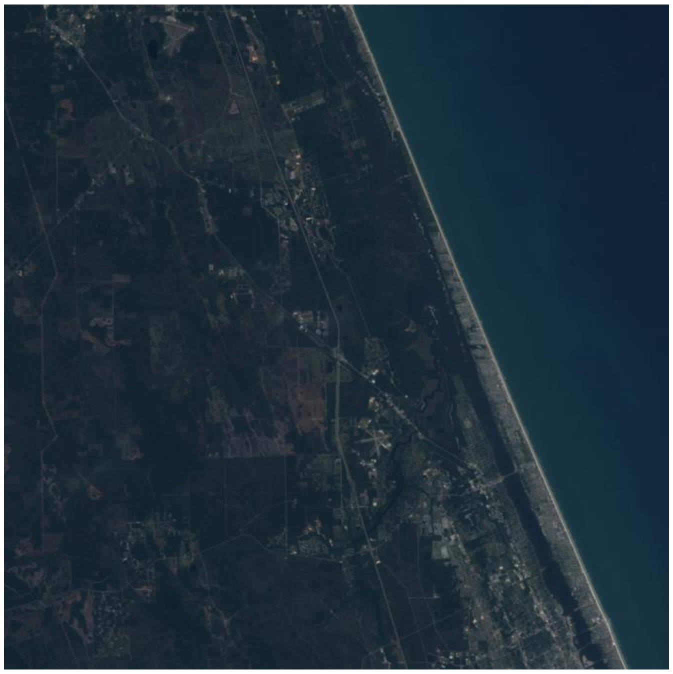
B.2 (CLOUD ENVIRONMENT MODEL)


CLOUD DEVELOPMENT ENVIRONMENT
Google Cloud Platform
Data collection/processing VM instance
Project data bucket
Predictino/analysis VM instance

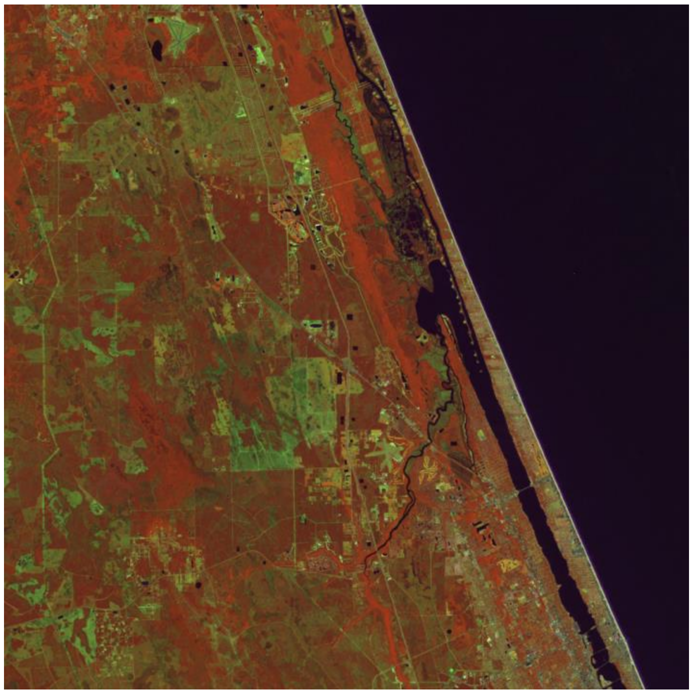
B.3 (ABSTRACT FACTORY IMPLEMENTATION)



B.4 (EXAMPLE LANDSAT IMAGES)



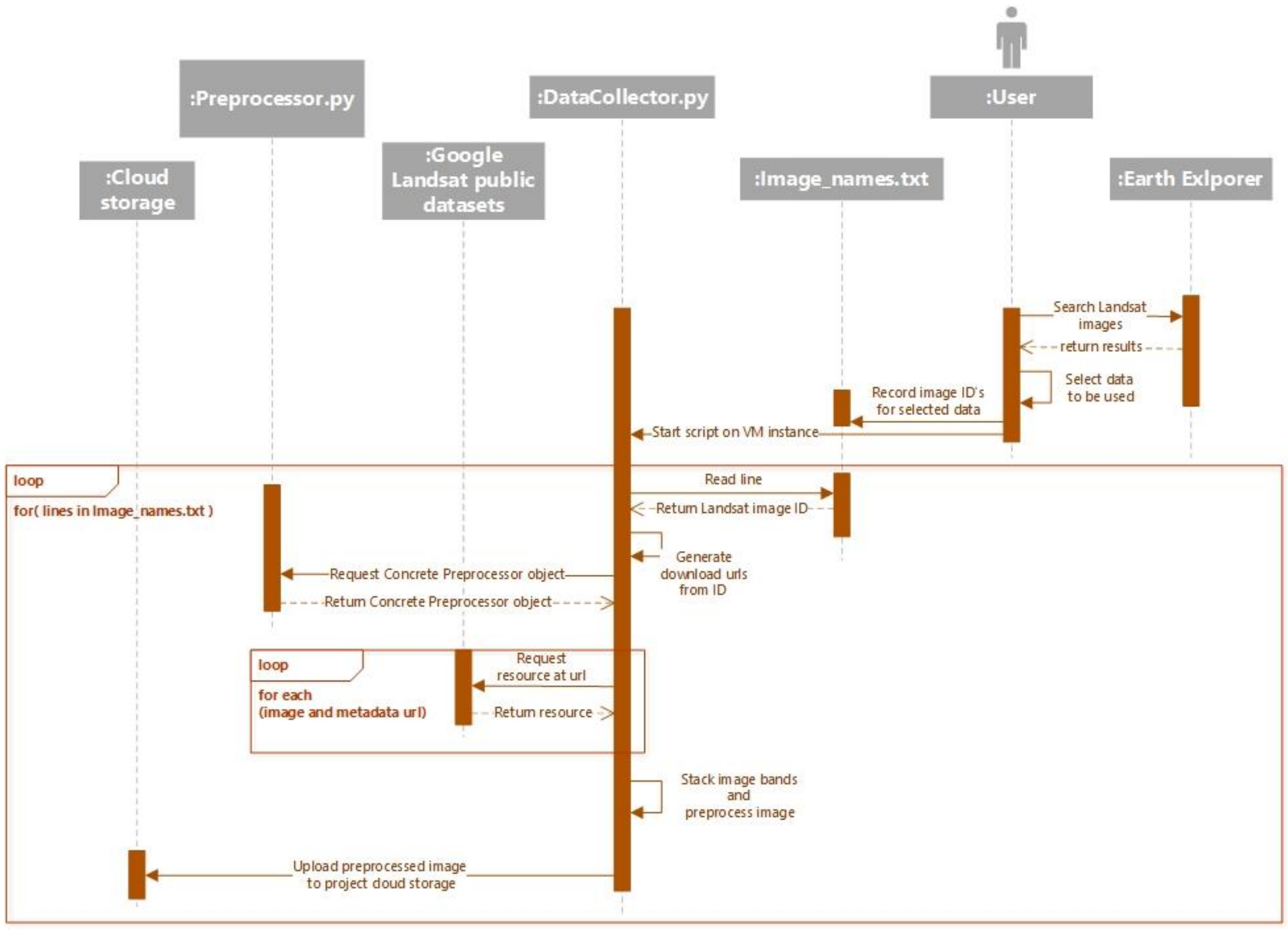
1. "True Color" (red, green, blue) image of Palm Coast, FL collected from Landsat 4, June 1992



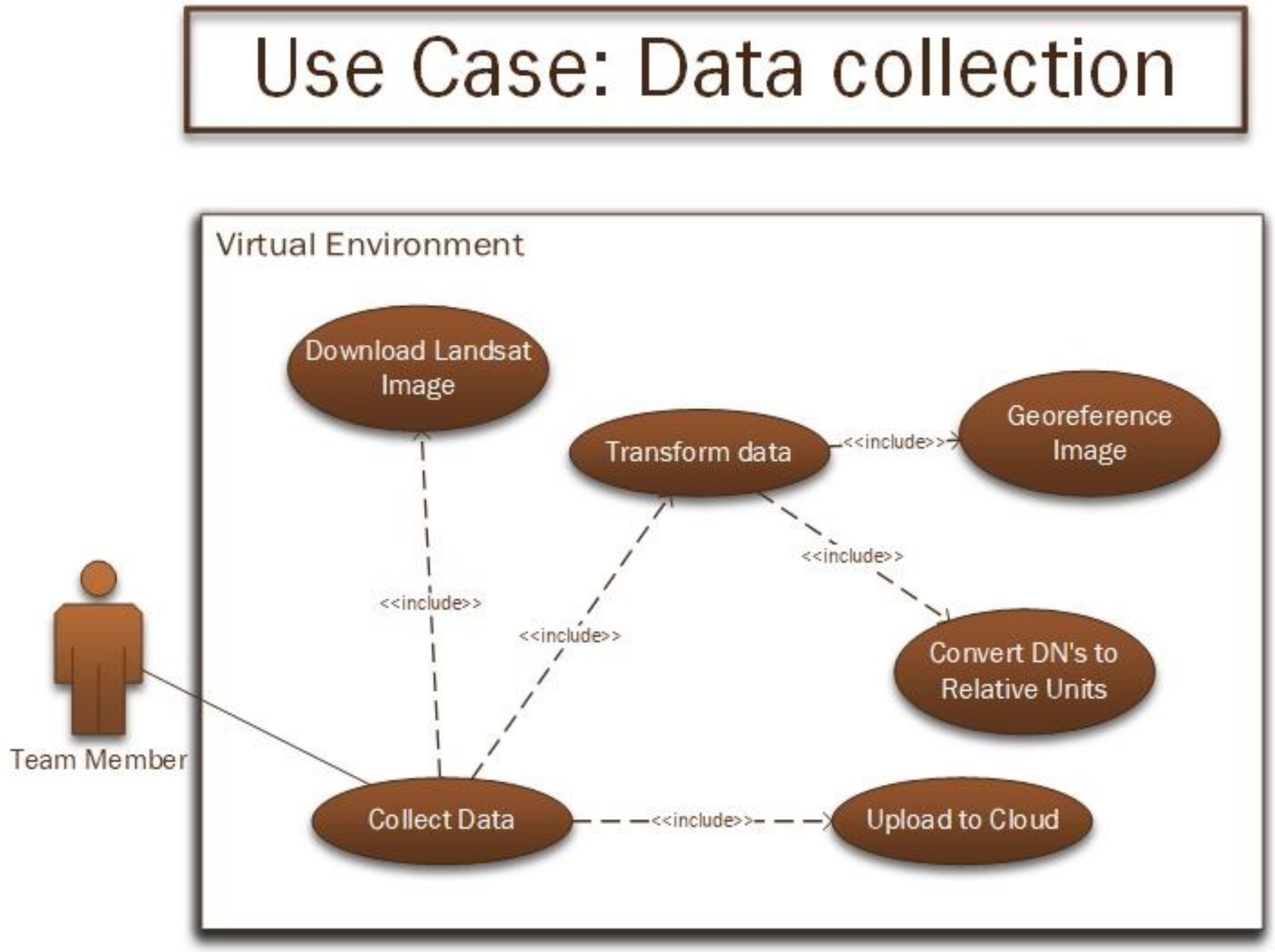
2 The same scene as above, but with near infrared, short-wave infrared and blue displayed as red, green and blue respectively

B.5 (UML DIAGRAMS)

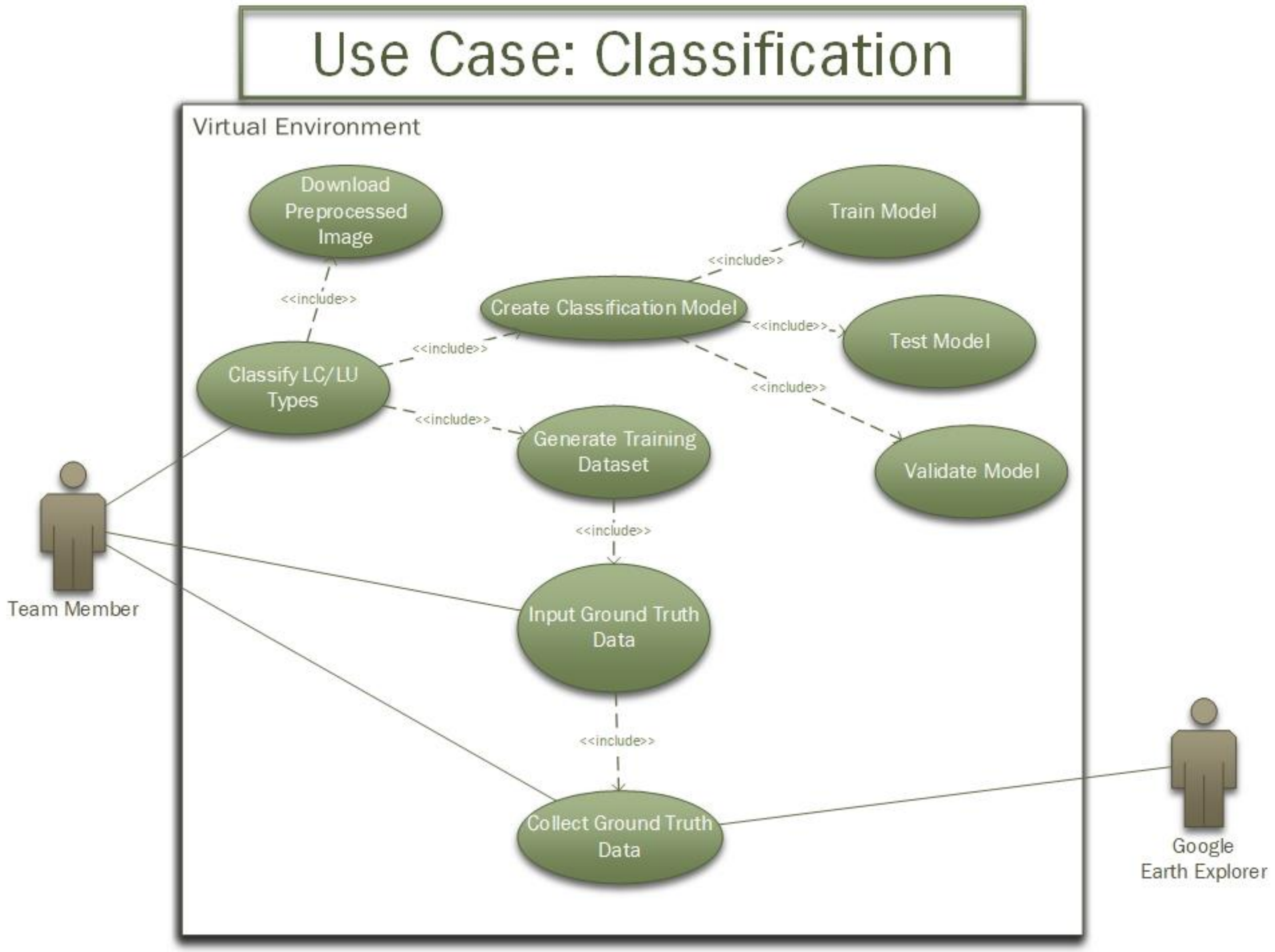
B.5.1 (DATA COLLECTION ACTIVITY DIAGRAM)


:Preprocessor.py
:DataCollector.py
:User
:Cloud storage
:Google Landsat public datasets
:Image_names.txt
:Earth Exlporer
Search Landsat images
return results
Select data to be used
Record image ID's for selected data
Start script on VM instance
loop
for(lines in Image_names.txt)
Read line
Return Landsat image ID
Generate download urls from ID
Request Concrete Preprocessor object
Return Concrete Preprocessor object
loop
for each (image and metadata url)
Request resource at url
Return resource
Stack image bands and preprocess image
Upload preprocessed image to project cloud storage

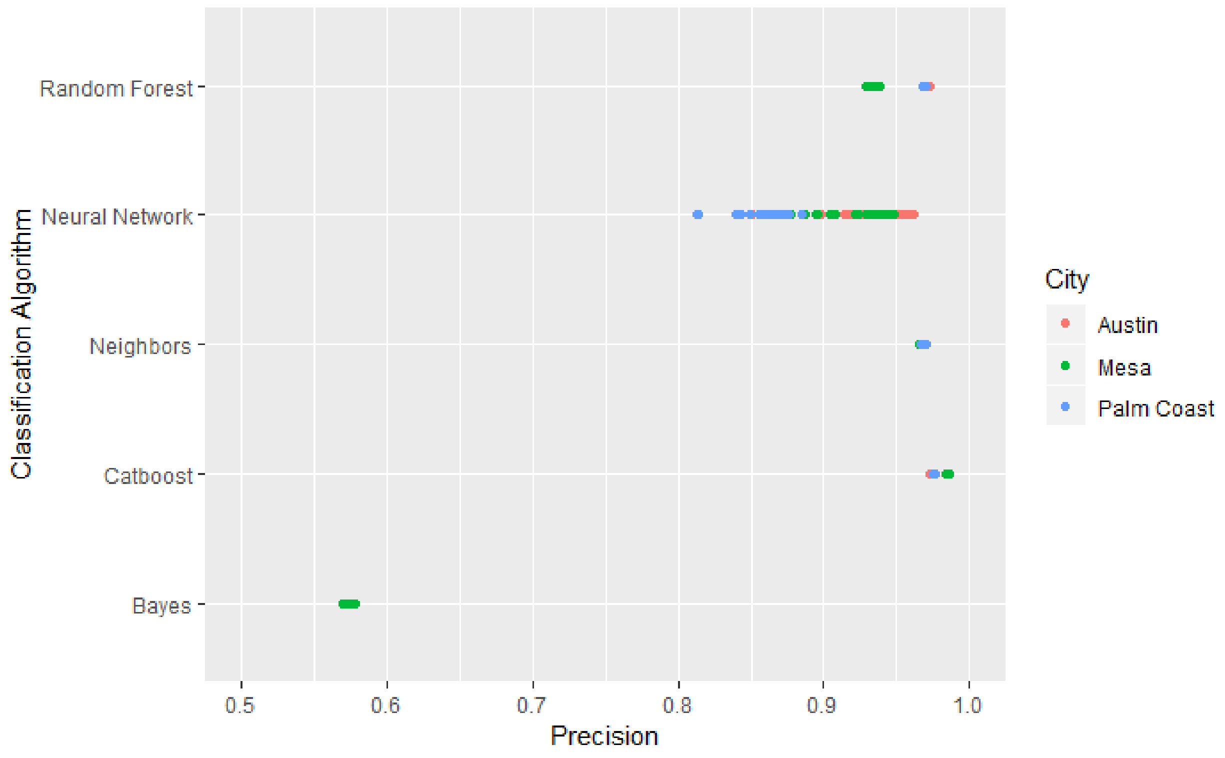
B.5.2 (USE CASE: DATA COLLECTION)

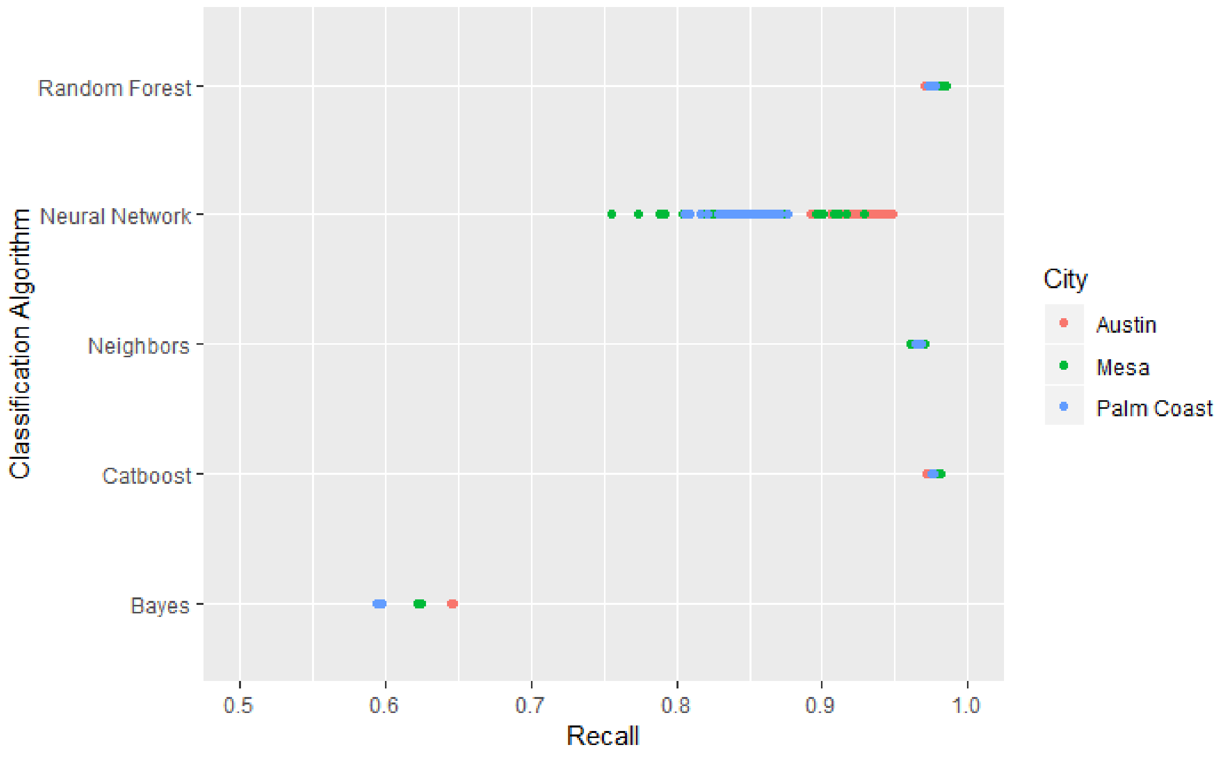


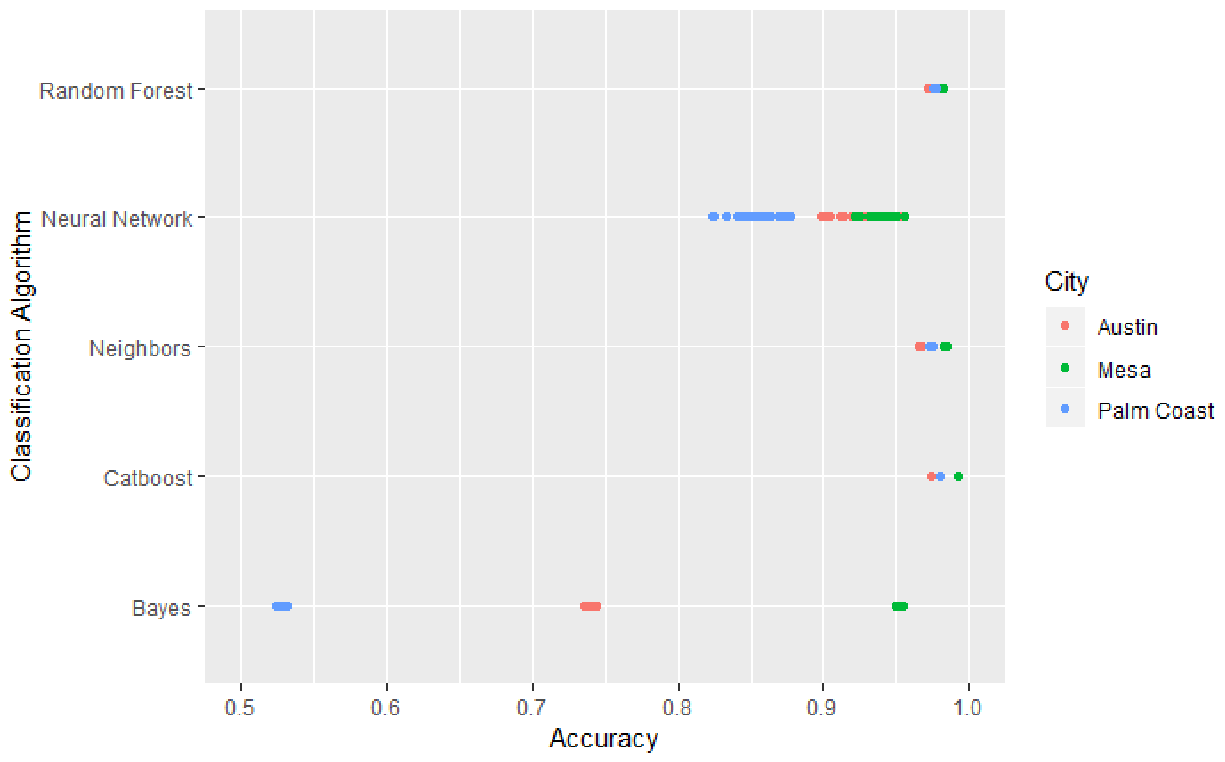
B.5.3 (USE CASE: CLASSIFICATION)



B.6 (PERFORMANCE OF MODELS)






Random Forest
Neural Network
Neighbors
Catboost
Bayes
Classification Algorithm
0.5
0.6
0.7
0.8
0.9
1.0
Accuracy
City
Austin
Mesa
Palm Coast

APPENDIX C (GENERATED CODE)

C.1 (DATA COLLECTION PACKAGE)

```
import gdal
import numpy as np
import Preprocessor as prep
from osgeo import gdal_array
base_url = "https://storage.googleapis.com/gcp-public-data-landsat/"
scene_names  = "addended.txt"
from google.cloud import storage
import wget
import os



client = storage.Client()
bucket = client.get_bucket('481_project')

'''
Parses the Landsat scene ID-string for [path#, row#, sensor#]
uses the data to construct the download urls for each band.
Landsat scenes are indexed by [path, row] numbers
'''
def string2data(scene_name):
    # Landsat satellite designation
    sat = scene_name[:4]
    # path number
    path_num = scene_name[10:13]
    # row number
    row_num = scene_name[13:16]
    # url for cloud storage location
    cons_url = "{}{}/01/{}/{}/{}/{}".format(base_url, sat, path_num,
row_num, scene_name, scene_name)
    bands = None
    # For landsat 4,5,7 (Landsat 6 was inoperable
    if int(sat[-1]) == 4 or int(sat[-1]) == 5:
        bands = ["{}_B1.TIF".format(cons_url),
"{}_B2.TIF".format(cons_url), "{}_B3.TIF".format(cons_url),
"{}_B4.TIF".format(cons_url), "{}_B5.TIF".format(cons_url),
"{}_B7.TIF".format(cons_url), "{}_B6.TIF".format(cons_url)]
    elif int(sat[-1]) == 7:
        bands = ["{}_B1.TIF".format(cons_url),
"{}_B2.TIF".format(cons_url), "{}_B3.TIF".format(cons_url),
                 "{}_B4.TIF".format(cons_url),
"{}_B5.TIF".format(cons_url), "{}_B7.TIF".format(cons_url),
                 "{}_B6_VCID_1.TIF".format(cons_url)]
    # For landsat 8
    else:
        bands = ["{}_B2.TIF".format(cons_url),
"{}_B3.TIF".format(cons_url), "{}_B4.TIF".format(cons_url),
                 "{}_B5.TIF".format(cons_url),
"{}_B6.TIF".format(cons_url), "{}_B7.TIF".format(cons_url),
                 "{}_B10.TIF".format(cons_url),
"{}_B11.TIF".format(cons_url)]
    return bands, int(sat[-1]), "{}_MTL.txt".format(cons_url)
```

```
def save2cloud(filename):
    blob = bucket.blob(filename)
    blob.upload_from_filename(filename)
    cmd = "rm {}".format(filename)
    os.system(cmd)

with open(scene_names ,"r") as src_file:
    for line in src_file:
        line = line.strip().split(",")
        out_file = line[1]
        bands, sat, meta_url = string2data(line[0])
        prepper = None
        print(sat)
        print(type(sat))
        if(sat==8):
            prepper = prep.Landsat8(sat, meta_url, bands)
        elif(sat==7):
            prepper = prep.Landsat7(sat, meta_url, bands)
        else:
            prepper = prep.Landsat45(sat, meta_url, bands)
        prepper.extract_metadata()
        prepper.collect()
        prepper.transform_data()
        prepper.geo_reference()
        np.save(out_file, prepper.data)
        save2cloud("{}.npy".format(line[1]))
        print("{}.npy saved to cloud\n\n".format(line[1]))
    src_file.close()
```

```python
import re
import math
import numpy as np
import urllib.request as req
from osgeo import gdal_array, gdal
from scipy import misc
import os
import wget
from numba import jit

solar_spectral_irradiances = {"B1": 1970,
                              "B2": 1842,
                              "B3": 1547,
                              "B4": 1044,
                              "B5": 225.7,
                              "B7": 82.06}

class AbstractPreprocessor(object):
    def __init__(self, satnum, meta_url, bands):
        self.satnum = satnum
        self.meta_url = meta_url
        self.bands = bands
        self.metadata = None
        self.data = None
        self.lats = {}
        self.longs = {}

    def collect(self, dtype='float32'):
        first = True
        arr = None
        j = 0
        for band in self.bands:
                j = j+1
                if first == True:
                    filename = wget.download(band)
                    arr = gdal_array.LoadFile(filename).astype(dtype)
                    cmd = "rm {}".format(filename)
                    os.system(cmd)
                    filename = None
                    print("\n\ndownloading {}\n\n\n".format(band))
                    first = False
                else:
                    print("\n\ndownloading {}\n\n\n".format(band))
                    filename = wget.download(band)
                    temp = gdal_array.LoadFile(filename).astype(dtype)
                    cmd = "rm {}".format(filename)
                    os.system(cmd)
                    filename = None
                    arr = np.dstack((arr, temp))

        self.data = arr
        arr = None

    def extract_metadata(self):
        pass


    def geo_reference(self):
        y_pixels = self.data.shape[0]
        x_pixels = self.data.shape[1]
        geo_layer = np.zeros((y_pixels,x_pixels), dtype="float32")
        geo_layer[0][0] = self.lats["UL"]
        geo_layer[1][1] = self.longs["UL"]
        geo_layer[0][-1] = self.lats["UR"]
        geo_layer[1][-2] = self.longs["UR"]
        geo_layer[-1][0] = self.lats["LL"]
        geo_layer[-2][1] = self.longs["LL"]
        geo_layer[-1][-1] = self.lats["LR"]
        geo_layer[-2][-2] = self.longs["LR"]
        self.data = np.dstack((self.data, geo_layer))

class Landsat8(AbstractPreprocessor):
    def __init__(self, satnum, meta_url, bands):
        AbstractPreprocessor.__init__(self, satnum, meta_url, bands)
        self.mult0 = {}
        self.add0 = {}
        self.mult1 = {}
        self.add1 = {}
        self.k1 = {}
        self.k2 = {}
        self.elv = None
    def extract_metadata(self, printing = False):
        print(self.meta_url)
        with req.urlopen(self.meta_url) as src:
            i = 0
            string = ""
            for line in src.readlines():
                if printing == True:
                    string = string + "{}:\n{}\n".format(i,
line.decode().strip())
                temp = line.decode().split()
                if(i==25):
                    self.lats['UL'] = float(temp[-1])
                elif(i==26):
                    self.longs['UL'] = float(temp[-1])
                elif (i == 27):
                    self.lats['UR'] = float(temp[-1])
                elif (i == 28):
                    self.longs['UR'] = float(temp[-1])
                elif (i == 29):
                    self.lats['LL'] = float(temp[-1])
                elif (i == 30):
                    self.longs['LL'] = float(temp[-1])
                elif (i == 31):
                    self.lats['LR'] = float(temp[-1])
                elif (i == 32):
                    self.longs['LR'] = float(temp[-1])
                elif(i==75):
                    self.elv  = float(temp[-1])
                elif(i==165):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult0['B1'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult0["B1"] = float(temp[0])*math.pow(10,
(float(temp[-1])))
                elif (i == 166):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult0['B2'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult0["B2"] = float(temp[0]) *math.pow(10,
(float(temp[-1])))
                elif (i == 167):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult0['B3'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult0["B3"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 168):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult0['B4'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult0["B4"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 169):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult0['B5'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult0["B5"] = float(temp[0]) *math.pow(10,
(float(temp[-1])))
                elif (i == 170):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult0['B6'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult0["B6"] = float(temp[0]) *math.pow(10,
(float(temp[-1])))
                elif (i == 171):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult0['B7'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult0["B7"] = float(temp[0]) *math.pow(10,
(float(temp[-1])))
                elif (i == 174):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult0['B10'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult0["B10"] = float(temp[0]) *math.pow(10,
(float(temp[-1])))
                elif (i == 175):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult0['B11'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult0["B11"] = float(temp[0]) *math.pow(10,
(float(temp[-1])))
                elif (i == 176):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add0['B1'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add0["B1"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 177):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add0['B2'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add0["B2"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 178):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add0['B3'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add0["B3"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 179):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add0['B4'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add0["B4"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 180):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add0['B5'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add0["B5"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 181):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add0['B6'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add0["B6"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 182):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add0['B7'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add0["B7"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 185):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add0['B10'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add0["B10"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 186):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add0['B11'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add0["B11"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))

                elif (i == 187):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult1['B1'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult1["B1"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 188):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult1['B2'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult1["B2"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
```

```
                elif (i == 189):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult1['B3'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult1["B3"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 190):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult1['B4'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult1["B4"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 191):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult1['B5'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult1["B5"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 192):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult1['B6'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult1["B6"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 193):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.mult1['B7'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.mult1["B7"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))

                elif (i == 196):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add1['B1'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add1["B1"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 197):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add1['B2'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add1["B2"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 198):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add1['B3'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add1["B3"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 199):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add1['B4'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add1["B4"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 200):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add1['B5'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add1["B5"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 201):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add1['B6'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add1["B6"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif (i == 202):
                    temp = temp[-1]
                    if (re.match(".+E.+", temp) == None):
                        self.add1['B7'] = float(temp)
                    else:
                        temp = temp.split("E")
                        self.add1["B7"] = float(temp[0]) * math.pow(10,
(float(temp[-1])))
                elif(i==207):
                    self.k1["B10"] = float(temp[-1])
                elif (i == 208):
                    self.k2["B10"] = float(temp[-1])

                elif (i == 209):
                    self.k1["B11"] = float(temp[-1])
                elif (i == 210):
                    self.k2["B11"] = float(temp[-1])
                i = i+1
                if(i>211):
                    break
        src.close()
        if printing==True:
            return string


    def transform_data(self):
        vis_mults0 = np.array([self.mult0["B2"], self.mult0["B3"],
self.mult0["B4"], self.mult0["B5"], self.mult0["B7"], self.mult0["B6"]])
        vis_mults0.shape = (1,1,6)
        vis_adds0 = np.array([self.add0["B2"], self.add0["B3"],
self.add0["B4"], self.add0["B5"], self.add0["B7"],self.add0["B6"]])
        vis_adds0.shape = (1, 1, 6)
        vis_mults1 = np.array([self.mult1["B2"], self.mult1["B3"],
self.mult1["B4"], self.mult1["B5"], self.mult1["B7"],
                               self.mult1["B6"]])
        vis_mults1.shape = (1, 1, 6)
        vis_adds1 = np.array(
            [self.add1["B2"], self.add1["B3"], self.add1["B4"],
self.add1["B5"], self.add1["B7"], self.add1["B6"]])
        vis_adds1.shape = (1, 1, 6)
        therm_mults = np.array([self.mult0["B10"], self.mult0["B11"]])
        therm_mults.shape = (1, 1, 2)
        therm_adds = np.array([self.add0["B10"], self.add0["B11"]])
        therm_adds.shape = (1, 1, 2)
        k1s  = np.array([self.k1["B10"], self.k1["B11"]])
        k1s.shape = (1, 1, 2)
        k2s = np.array([self.k2["B10"], self.k2["B11"]])
        k2s.shape = (1, 1, 2)
        vis = self.data[:,:,:6]
        therm = self.data[:,:,6:]
        self.data = None
        vis = np.multiply(vis, vis_mults0)
        vis = np.add(vis, vis_adds0)
        vis = np.multiply(vis, vis_mults1)
        vis = np.add(vis, vis_adds1)
        vis = np.abs(np.multiply(vis, (1/math.sin(self.elv))))
        therm = np.multiply(therm, therm_mults)
        therm = np.add(therm, therm_adds)
        therm = np.reciprocal(therm)
        therm = np.multiply(k1s, therm)
        therm = np.add(1, therm)
        therm = np.multiply(k2s, np.reciprocal(np.log(therm)))
        self.data = np.dstack((vis, therm))



class Landsat7(AbstractPreprocessor):

    def __init__(self, satnum, meta_url, bands):
        AbstractPreprocessor.__init__(self, satnum, meta_url, bands)
        self.earth2sun_distance = None
        self.gains_table = {}
        self.k1 = 666.09
        self.k2  = 1282.71
        self.solar_zenith = None
        self.lmax_lmin_table = {}
        irradiance =np.array([1970,1842,1547,1044,225.7,82.06])
        irradiance.shape = (1,1,6)
        self.solar_spectral_irradiances = irradiance

    def extract_metadata(self, printing = False):
        with req.urlopen(self.meta_url) as src:
            i = 0
            string = ""
            for line in src.readlines():
                if printing == True:
                    string = string + "{}:\n{}\n".format(i,
line.decode().strip())
                temp = line.decode().split()
                if(i==25):
                    self.lats['UL'] = float(temp[-1])
                elif(i==26):
                    self.longs['UL'] = float(temp[-1])
                elif (i == 27):
                    self.lats['UR'] = float(temp[-1])
                elif (i == 28):
                    self.longs['UR'] = float(temp[-1])
                elif (i == 29):
                    self.lats['LL'] = float(temp[-1])
                elif (i == 30):
                    self.longs['LL'] = float(temp[-1])
                elif (i == 31):
                    self.lats['LR'] = float(temp[-1])
                elif (i == 32):
                    self.longs['LR'] = float(temp[-1])
                elif (i == 67):
                    self.solar_zenith = 90-float(temp[-1])
                elif(i==68):
                    self.earth2sun_distance  = float(temp[-1])
                elif(i==85):
                    self.lmax_lmin_table['B1'] = {'max':float(temp[-1])}
                elif (i == 86):
                    self.lmax_lmin_table['B1']['min'] = float(temp[-
1].strip('-'))
                elif (i == 87):
                    self.lmax_lmin_table['B2'] = {'max': float(temp[-1])}
                elif (i == 88):
                    self.lmax_lmin_table['B2']['min'] = float(temp[-
1].strip('-'))
                elif (i == 89):
                    self.lmax_lmin_table['B3'] = {'max': float(temp[-1])}
                elif (i == 90):
                    self.lmax_lmin_table['B3']['min'] = float(temp[-
1].strip('-'))
                elif (i == 91):
                    self.lmax_lmin_table['B4'] = {'max': float(temp[-1])}
                elif (i == 92):
                    self.lmax_lmin_table['B4']['min'] = float(temp[-
1].strip('-'))
                elif (i == 93):
                    self.lmax_lmin_table['B5'] = {'max': float(temp[-1])}
                elif (i == 94):
                    self.lmax_lmin_table['B5']['min'] = float(temp[-
1].strip('-'))
                elif (i == 97):
                    self.lmax_lmin_table['B6'] = {'max': float(temp[-1])}
                elif (i == 98):
                    self.lmax_lmin_table['B6']['min'] = float(temp[-
1].strip('-'))
                elif (i == 99):
                    self.lmax_lmin_table['B7'] = {'max': float(temp[-1])}
                elif (i == 100):
                    self.lmax_lmin_table['B7']['min'] = float(temp[-
1].strip('-'))
                i = i+1
                if(i>101):
                    break
        src.close()
        if printing == True:
            return string


    def transform_data(self):
        self.data = np.subtract(self.data, 1.00001)
        bands = ['B1', 'B2', 'B3', 'B4', 'B5', 'B7', 'B6']
        mults = []
        adds = []
        for band in bands:
            mults.append(((self.lmax_lmin_table[band]['max'] -
self.lmax_lmin_table[band]['min']) / 254))
            adds.append(self.lmax_lmin_table[band]['min'])
        mults = np.array(mults)
        mults.shape = (1, 1, 7)
        adds = np.array(adds)
        adds.shape = (1, 1, 7)
        self.data = np.multiply(self.data, mults)
        self.data = np.add(self.data, adds)
        vis, therm = self.data[:, :, 0:6], self.data[:, :, 6:]
        therm = np.add(therm, 1.00001)
        self.data = None
        solar_scalar = (math.pi * self.earth2sun_distance) /
math.cos(self.solar_zenith)
        vis = np.multiply(np.true_divide(vis,
self.solar_spectral_irradiances), solar_scalar)
        therm = np.reciprocal(therm)
        therm = np.multiply(self.k1, therm)
        therm = np.add(therm, 1)
        therm = np.log(therm)
        therm = np.reciprocal(therm)
        therm = np.multiply(therm, self.k2)
        self.data = np.dstack((vis, therm))
        vis = None
        therm = None



class Landsat45(AbstractPreprocessor):
    def __init__(self, satnum, meta_url, bands):
        AbstractPreprocessor.__init__(self, satnum, meta_url, bands)
        self.earth2sun_distance = None
        self.gains_table = {}
        self.solar_zenith = None
        self.lmax_lmin_table = {}
        self.irradiance = None
        if(satnum==4):
            self.irradiance  =[1958,1826,1554,1033,214.7,80.70]
            self.k1 = 671.62
            self.k2 = 1284.30
        else:
            self.irradiance = [1958, 1827, 1551, 1036, 214.9, 80.65]
            self.k1 = 607.76
            self.k2 = 1260.56
        self.irradiance = np.array(self.irradiance)
        self.irradiance.shape = (1, 1, 6)
        self.solar_spectral_irradiances = self.irradiance

    def extract_metadata(self, printing = False):
        with req.urlopen(self.meta_url) as src:
            i = 0
            string = ""
            for line in src.readlines():
                if printing == True:
                    string = string + "{}:\n{}\n".format(i,
line.decode().strip())
                temp = line.decode().split()
                if(i==26):
                    self.lats['UL'] = float(temp[-1])
                elif(i==27):
                    self.longs['UL'] = float(temp[-1])
                elif (i == 28):
                    self.lats['UR'] = float(temp[-1])
                elif (i == 29):
                    self.longs['UR'] = float(temp[-1])
                elif (i == 30):
                    self.lats['LL'] = float(temp[-1])
                elif (i == 31):
                    self.longs['LL'] = float(temp[-1])
                elif (i == 32):
                    self.lats['LR'] = float(temp[-1])
                elif (i == 33):
                    self.longs['LR'] = float(temp[-1])
                elif (i == 66):
                    self.solar_zenith = 90-float(temp[-1])
                elif(i==67):
                    self.earth2sun_distance  = float(temp[-1])
                elif(i==88):
                    self.lmax_lmin_table['B1'] = {'max':float(temp[-1])}
                elif (i == 89):
                    self.lmax_lmin_table['B1']['min'] = float(temp[-
1].strip('-'))
                elif (i ==90):
                    self.lmax_lmin_table['B2'] = {'max': float(temp[-1])}
                elif (i == 91):
                    self.lmax_lmin_table['B2']['min'] = float(temp[-
1].strip('-'))
                elif (i == 92):
                    self.lmax_lmin_table['B3'] = {'max': float(temp[-1])}
                elif (i == 93):
                    self.lmax_lmin_table['B3']['min'] = float(temp[-
1].strip('-'))
                elif (i == 94):
                    self.lmax_lmin_table['B4'] = {'max': float(temp[-1])}
                elif (i == 95):
                    self.lmax_lmin_table['B4']['min'] = float(temp[-
1].strip('-'))
                elif (i == 96):
                    self.lmax_lmin_table['B5'] = {'max': float(temp[-1])}
                elif (i == 97):
                    self.lmax_lmin_table['B5']['min'] = float(temp[-
1].strip('-'))
                elif (i == 98):
                    self.lmax_lmin_table['B6'] = {'max': float(temp[-1])}
                elif (i == 99):
                    self.lmax_lmin_table['B6']['min'] = float(temp[-
1].strip('-'))
                elif (i == 100):
                    self.lmax_lmin_table['B7'] = {'max': float(temp[-1])}
                elif (i == 101):
                    self.lmax_lmin_table['B7']['min'] = float(temp[-
1].strip('-'))
                i = i+1
                if(i>101):
                    src.close()
                    if printing == True:
                        return string
                    break


    def transform_data(self):
        self.data = np.subtract(self.data, 1.00001)
        bands = ['B1','B2','B3','B4','B5','B7', 'B6']
        mults  = []
        adds = []
        for band in bands:
            mults.append(((self.lmax_lmin_table[band]['max'] -
self.lmax_lmin_table[band]['min'])/254))
            adds.append(self.lmax_lmin_table[band]['min'])
        mults = np.array(mults)
        mults.shape = (1,1,7)
        adds = np.array(adds)
        adds.shape = (1,1,7)
        self.data = np.multiply(self.data, mults)
        self.data = np.add(self.data ,adds)
        vis, therm = self.data[:, :, 0:6], self.data[:, :, 6:]
        therm = np.add(therm, 1.00001)
        self.data = None
        solar_scalar =
(math.pi*self.earth2sun_distance)/math.cos(self.solar_zenith)
        vis = np.multiply(np.true_divide(vis,
self.solar_spectral_irradiances), solar_scalar)
        therm  = np.reciprocal(therm)
        therm = np.multiply(self.k1, therm)
        therm = np.add(therm, 1)
        therm = np.log(therm)
        therm = np.reciprocal(therm)
        therm = np.multiply(therm, self.k2)
        self.data = np.dstack((vis, therm))
        vis = None
        therm = None
```

```
import sklearn as sk
from sklearn.externals import joblib
from sklearn.ensemble import RandomForestClassifier
from google.cloud import storage
import numpy as np
import os
import threading
from pomegranate import *
from time import sleep
import copy
import json
from scipy.misc import toimage
from catboost import CatBoostClassifier, Pool

class Converter:
     def __init__(self, model,  threaded=False):
                 self.geo_ref_points = [[30.69611111, -
98.47833333],[29.41027778, -81.69722222], [33.55388889, -113.08194444]]
                 self.client = storage.Client()
                 self.bucket = self.client.get_bucket('481_project')
                 ###   load the classificatino model
                 #model = open(model)
                 #model = json.load(model)
                 #self.classifier = NaiveBayes.from_json(model)
                 self.slices = []
                 self.processed_slices = {}
                 self.data = None
                 self.transform_matrix = None
                 self.threaded = threaded

     '''
     def save2cloud(self, filename):

                 #Save Class data
                 np.save(filename, self.data)
                 blob = self.bucket.blob("LCLU/Bayes/" + filename)
                 blob.upload_from_filename(filename)
                 cmd = "rm {}".format(filename)
                 os.system(cmd)

                 #Save LCLU map
                 png_filename = filename.replace(".npy", ".png")
                 return_arr = np.zeros(shape=(self.data.shape[0],
self.data.shape[1], 3))
                 for i in range(self.data.shape[0]):
                           for j in range(self.data.shape[1]):
                                k = int(self.data[i, j] - 1)
                                return_arr[i, j, k] = 250
                 im = toimage(return_arr)
                 im.save(png_filename)
                 blob = self.bucket.blob("LCLU/Bayes/images/" +
png_filename)
                 blob.upload_from_filename(png_filename)
                 cmd = "rm {}".format(png_filename)
```

```
                os.system(cmd)

        def load_data(self, data):
                if self.threaded:
                        for i in range(2):
                            start = i*1250
                            stop = (i+1)*1250
                            if i <1:

        self.slices.append(data[start:stop, :, :])
                            else:

        self.slices.append(data[start:, :, :])
                else:
                        self.slices.append(data)

        def convert(self):
                if self.threaded:
                        threads_list = []
                        i = 0
                        still_active = True
                        for slice in self.slices:
                            thread =
threading.Thread(target=self.classify_slice, args=(slice,i))
                            print("\n\nWWWWWWWWWWWWWWWWWW\nStarting
slice{}".format(i))
                            thread.start()
                            threads_list.append(thread)
                            i = i+1
                        #loop until threads have terminated
                        while(still_active):
                            i = 0
                            for thread in threads_list:
                                    if thread.is_alive():
                                            i = i+1
                            if i == 0:
                                    still_active = False
                            else:
                                    print("\n\n----------{}
Threads Still Active---------\n\n".format(i))
                                    sleep(20)
                        print("\n\n00000 Fniished Converting Slices
00000\n\n")
                        self.slices = None
                        threads_list = None
                else:
                        print("converting")
                        self.classify_slice(self.slices[0], 0)

        def stack_proc_slices(self):
                if self.threaded:
                        temp_data = self.processed_slices["0"]
                        for i in range(1,2):
                              temp_data = np.vstack((temp_data,
self.processed_slices["{}".format(i)]))
                         self.data = temp_data
                         self.processed_slices = None
               else:
                         self.data = self.processed_slices["0"]

     def classify_slice(self, slice, index):
               points = []
               for i in range(slice.shape[0]):
                         for j in range(slice.shape[1]):
                              points.append(slice[i,j,:])

     print("\n\n\n_______Processing_________\nSlice{}".format(index))
               proc_slice = self.classifier.predict(points)
               proc_slice.shape = (slice.shape[0],slice.shape[1])
               self.processed_slices["{}".format(index)] = proc_slice
               print("\n\nTTTTTTTTTTTT\nslice {}
processed".format(index))
               '''
     def geo_ref(self, geo_layer):
               y_pixels = int(geo_layer.shape[0])
               x_pixels = int(geo_layer.shape[1])
               b = np.array(([1, 1, geo_layer[0][0], geo_layer[1][1]],
                                          [1, 1, geo_layer[0][-1],
geo_layer[1][-2]],
                                          [1, 1, geo_layer[-1][0],
geo_layer[-2][1]],
                                          [1, 1, geo_layer[-1][-1],
geo_layer[-2][-2]]))
               a = np.array([[1, 1, 0, 0],  # coefficient matrix
                                          [1, 1, 0, x_pixels],
                                          [1, 1, y_pixels, 0],
                                          [1, 1, y_pixels,
x_pixels]])
               x = np.linalg.lstsq(a, b, rcond=-1)[0]
               x = np.linalg.pinv(x)
               self.transform_matrix = x

     '''
               This method uses the previously computed transformation
matrix to find pixels corresponding to lat/long coordinates
     '''

     def get_pixel_values(self, lat, long):
               b = np.array([1, 1, lat, long])
               a = np.dot(b, self.transform_matrix)
               y = int(a[-2])
               x = int(a[-1])
               return y, x
```

```
import os
import numpy as np
import sklearn as sk
import csv
from google.cloud import storage
import re
from matplotlib.path import Path


class PolyHandler:
      def __init__(self):
                  samples_dir = "/home/lizard/483_Landsat_project/samples/"
                  self.client = storage.Client()
                  self.bucket = self.client.get_bucket('481_project')
                  # points_lists are the csv files with hand collected
points and classes
                  self.points_lists = ["{}A_polys.csv".format(samples_dir)]
                  # files_list are text files containing the names of the
files for each image set
                  self.files_list =
["{}A_poly_names.txt".format(samples_dir)]
                  self.dst_files =
["{}A_poly_samples.npy".format(samples_dir)]
                  self.data = None
                  self.transform_matrix = None

      """
      This method compiles the hand collected points into a set of sample
observations
      There are around (50 * #images * 3) samples.
      Each sample is the vector: [band_1 reflectance, band_2 reflectance,
..., class]

      For each geographic site, the method compiles the hand collected
data points into a list:
                  Then for each image data object corresponding to each
site:
                              for each ground-truth point in the list:
                                    samples the spectral reflectance values
at the ground-truth lat/long
      """

      def load_data(self):
                  samples = []
                  for i in range(1):
                              data_entries = []
                              pt_list = self.points_lists[i]
                              nm_list = self.files_list[i]
                              with open(pt_list, newline='', encoding='utf-
8') as csv_file:
                                    csv_reader = csv.reader(csv_file)
                                    for row in csv_reader:
                                        point = [float(re.sub("[^0-
9.\-]", "", row[0])), float(re.sub("[^0-9.\-]", "", row[1])),
float(re.sub("[^0-9.\-]", "", row[2])),

float(re.sub("[^0-9.\-]", "", row[3])), float(re.sub("[^0-9.\-]", "",
row[4])), float(re.sub("[^0-9.\-]", "", row[5])),

float(re.sub("[^0-9.\-]", "", row[6])), float(re.sub("[^0-9.\-]", "",
row[7])),

int(re.sub("[^0-9.\-]", "", row[8]))]
                                        data_entries.append(point)
                              csv_file.close()
                         with open(nm_list, "r") as names_file:
                              for line in names_file:
                                        line = line.strip()
                                        print("\nprocessing:
\t{}\n".format(line))
                                        blob =
self.bucket.blob(line)

     blob.download_to_filename(line)
                                        raster = np.load(line)
                                        y_max  = raster.shape[0]
                                        geo = raster[:, :, -1]
                                        spectral_layers =
np.abs(raster[:, :, :-1])
                                        if (spectral_layers.shape[2]
> 7):
                                                  vis, therm_1,
therm_2 = spectral_layers[:, :, :-2], spectral_layers[:, :, -2],
spectral_layers[:, :, -1]
                                                  spectral_layers =
None
                                                  therm =
np.add(therm_1, therm_2)
                                                  therm =
np.divide(therm, 2)
                                                  spectral_layers =
np.dstack((vis, therm))
                                                  vis = None
                                                  therm_1 = None
                                                  therm_2 = None
                                        self.geo_ref(geo)
                                        raster = None
                                        cmd = "rm {}".format(line)
                                        os.system(cmd)
                                        samples_as_array = None
                                        for point in data_entries:
                                                  y1, x1 =
self.get_pixel_values(point[0], point[1])
                                                  y2, x2 =
self.get_pixel_values(point[2], point[3])
```

```
                                                            y3, x3 =
self.get_pixel_values(point[4], point[5])
                                                            y4, x4 =
self.get_pixel_values(point[6], point[7])
                                                            label = point[-1]
                                                            verts = [(x1,
y1), (x2, y2), (x3, y3), (x4, y4), (x1, y1)]
                                                            codes =
[Path.MOVETO, Path.LINETO, Path.LINETO, Path.LINETO, Path.CLOSEPOLY]
                                                            path =
Path(verts, codes)
                                                            extents =
path.get_extents().get_points()
                                                            print(extents)
                                                            for y in
range(int(extents[0, 1]), int(extents[1, 1])):
                                                                  for x in
range(int(extents[0, 0]), int(extents[1, 0])):

      if path.contains_point((x,y)):

            sample = np.ravel(spectral_layers[y, x, :])

            sample = np.append(sample, label)

            samples.append(sample)
                                                            if
samples_as_array is None:

      samples_as_array = np.ravel(np.array(samples[0])).astype("float32")
                                                            for j in range(1,
len(samples) - 1):
                                                                  samp =
np.ravel(np.array(samples[j]))

      samples_as_array = np.dstack((samp, samples_as_array))
                                                            samples = []

      print("\n{}\n".format(samples_as_array.shape))
                                    samples = []
                                    np.save(self.dst_files[i],
samples_as_array.T)
                                    names_file.close()

      """
      This method uses linear algebra to find the transformation matrix
      to convert points from lat/long coordinates to x/y pixel values.
      The Upper/Lower--Left/Right latitude and longitude values for the
corner pixels of each image were obtained from landsat metadata.
      The latitude and longitude values for the corners are stored in the
geospatial layer of the numpy data cube

      The method sets the instance variable: "self.transform_matrix"
      This variable is used in the get_pixel_values method to convert from
lat/long to pixel coordinates.

      solves x^-1 for Ax=b where A=[corner pixel coordinates] and b =
[corner lat/long values]
      """

      def geo_ref(self, geo_layer):
                y_pixels = int(geo_layer.shape[0])
                x_pixels = int(geo_layer.shape[1])
                b = np.array(([1, 1, geo_layer[0][0], geo_layer[1][1]],
                                          [1, 1, geo_layer[0][-1],
geo_layer[1][-2]],
                                          [1, 1, geo_layer[-1][0],
geo_layer[-2][1]],
                                          [1, 1, geo_layer[-1][-1],
geo_layer[-2][-2]]))
                a = np.array([[1, 1, 0, 0],  # coefficient matrix
                                          [1, 1, 0, x_pixels],
                                          [1, 1, y_pixels, 0],
                                          [1, 1, y_pixels,
x_pixels]])
                x = np.linalg.lstsq(a, b, rcond=-1)[0]
                x = np.linalg.pinv(x)
                self.transform_matrix = x

      '''
      This method uses the previously computed transformation matrix to
find pixels corresponding to lat/long coordinates
      '''

      def get_pixel_values(self, lat, long):
                b = np.array([1, 1, lat, long])
                a = np.dot(b, self.transform_matrix)
                y = int(a[-2])
                x = int(a[-1])
                return y, x

      def shuffle(self):
                self.data = np.random.shuffle(self.data)

      def cleave(self):
                sample_reflectance_values = (self.data[:, :-1])
                sample_class_label = ((self.data[:, -1])).astype(int)
                return sample_reflectance_values, sample_class_label
```

C.2 (STATISTICAL ANALYSIS)

```
require(ggplot2)
library(car)
#read in datasets
A_Bayes <- read.csv("A_Bayes.csv")
A_Cat <- read.csv("A_Cat.csv")
A_Neighbors <- read.csv("A_Neighbors.csv")
A_NN <- read.csv("A_NN.csv")
A_RF <- read.csv("A_RF.csv")
M_Bayes <- read.csv("M_Bayes.csv")
M_Cat <- read.csv("M_Cat.csv")
M_Neighbors <- read.csv("M_Neighbors.csv")
M_NN <- read.csv("M_NN.csv")
M_RF <- read.csv("M_RF.csv")
PC_Bayes <- read.csv("PC_Bayes.csv")
PC_Cat <- read.csv("PC_Cat.csv")
PC_Neighbors <- read.csv("PC_Neighbors.csv")
PC_NN <- read.csv("PC_NN.csv")
PC_RF <- read.csv("PC_RF.csv")
#create new data frame for each metric
accuracy.df = data.frame(City = c(rep("Austin",150), rep("Mesa",150),
rep("Palm Coast",150)),
                    Algorithm = c(rep(c(rep("Bayes",30),
rep("Catboost",30), rep("Neighbors",30), rep("Neural Network",30),
rep("Random Forest",30)),3)),
                    Accuracy = c(A_Bayes$Accuracy,
A_Cat$Accuracy[seq(1,30)], A_Neighbors$Accuracy, A_NN$Accuracy,
A_RF$Accuracy[seq(1,30)],
                              M_Bayes$Accuracy, M_Cat$Accuracy[seq(1,30)],
M_Neighbors$Accuracy, M_NN$Accuracy, M_RF$Accuracy[seq(1,30)],
                              PC_Bayes$Accuracy,
PC_Cat$Accuracy[seq(1,30)], PC_Neighbors$Accuracy, PC_NN$Accuracy,
PC_RF$Accuracy[seq(1,30)]))
precision.df = data.frame(City = c(rep("Austin",150), rep("Mesa",150),
rep("Palm Coast",150)),
                          Algorithm = c(rep(c(rep("Bayes",30),
rep("Catboost",30), rep("Neighbors",30), rep("Neural Network",30),
rep("Random Forest",30)),3)),
                          Precision = c(A_Bayes$Precision,
A_Cat$Precision[seq(1,30)], A_Neighbors$Precision, A_NN$Precision,
A_RF$Precision[seq(1,30)],
                                        M_Bayes$Precision,
M_Cat$Precision[seq(1,30)], M_Neighbors$Precision, M_NN$Precision,
M_RF$Precision[seq(1,30)],
                                        PC_Bayes$Precision,
PC_Cat$Precision[seq(1,30)], PC_Neighbors$Precision, PC_NN$Precision,
PC_RF$Precision[seq(1,30)]))
recall.df = data.frame(City = c(rep("Austin",150), rep("Mesa",150),
rep("Palm Coast",150)),
                       Algorithm = c(rep(c(rep("Bayes",30),
rep("Catboost",30), rep("Neighbors",30), rep("Neural Network",30),
rep("Random Forest",30)),3)),
                       Recall = c(A_Bayes$Recall, A_Cat$Recall[seq(1,30)],
A_Neighbors$Recall, A_NN$Recall, A_RF$Recall[seq(1,30)],
                                M_Bayes$Recall, M_Cat$Recall[seq(1,30)],
M_Neighbors$Recall, M_NN$Recall, M_RF$Recall[seq(1,30)],
                                PC_Bayes$Recall, PC_Cat$Recall[seq(1,30)],
PC_Neighbors$Recall, PC_NN$Recall, PC_RF$Recall[seq(1,30)]))

#data visualization
ggplot(accuracy.df, aes(Accuracy, Algorithm, colour = City)) +
geom_point()
ggplot(precision.df, aes(Precision, Algorithm, colour = City)) +
geom_point()
ggplot(recall.df, aes(Recall, Algorithm, colour = City)) + geom_point()
#models
accuracy.model <- aov(Accuracy ~ City*Algorithm, data = accuracy.df)
precision.model <- aov(Precision ~ City*Algorithm, data = precision.df)
recall.model <- aov(Recall ~ City*Algorithm, data = recall.df)
#analysis of overall model
summary(accuracy.model)
summary(precision.model)
summary(recall.model)
#residual plots and assumptions
#1. Homogoneity of variance - Violated
plot(accuracy.model,1)
plot(precision.model,1)
plot(recall.model,1)
leveneTest(Accuracy ~ City*Algorithm, data = accuracy.df)
leveneTest(Precision ~ City*Algorithm, data = precision.df)
leveneTest(Recall ~ City*Algorithm, data = recall.df)
#2. Normality - Violated
plot(accuracy.model,2)
plot(precision.model,2)
plot(recall.model,2)
acc.res <- residuals(object = accuracy.model)
pre.res <- residuals(object = precision.model)
rec.res <- residuals(object = recall.model)
shapiro.test(x = acc.res)
shapiro.test(x = pre.res)
shapiro.test(x = rec.res)
###Because both assumptions are violated, we will instead use
bootstrapping,
###as transformations didn't work either (see bottom of code)

library(boot)
set.seed(03182019)
#Accuracy
n <- 30*2 #simulating taking two samples of 30, so times 2
B <- 10000 #number of resamples we'll run
Cities <- c("Austin","Mesa","Palm_Coast")
Algorithms <-
c("Bayes","Catboost","Neighbors","Neural_Network","Random_Forest")
veclist <- do.call(paste,expand.grid(Cities,Algorithms)) #get every combo
of city and algo
count <- 1
p.values <- rep(-1,105)
for(i in 1:15){
  temp <- strsplit(veclist[i], " ")[[1]]
  name1 <- gsub("_", " ", temp[1]) #City1
  name2 <- gsub("_", " ", temp[2]) #Algo1
  for(j in (i+1):15){
    temp2 <- strsplit(veclist[j], " ")[[1]]
    name3 <- gsub("_", " ", temp2[1]) #City2
    name4 <- gsub("_", " ", temp2[2]) #Algo2
    if( i != 15){ #prevent last two comparisons, so theres ((15-1)*15) / 2
= 105 total comparisons
      variable1 <- accuracy.df$Accuracy[accuracy.df$City == name1 &
accuracy.df$Algorithm == name2]
      variable2 <- accuracy.df$Accuracy[accuracy.df$City == name3 &
accuracy.df$Algorithm == name4]
      variable <- c(variable1, variable2)
      #Analysis
      test.stat <- abs(mean(variable1) - mean(variable2)) #test-statistic
      B.samples <- matrix(sample(variable, size = n*B, replace=TRUE),
#10,000 resamples with replacement
                            nrow=n, ncol=B)
      Boot.test.stat <- rep(0,B)
      for(i in 1:B){ #for each sample, find new test-statistic
        Boot.test.stat[i] <- abs(mean(B.samples[1:30,i]) -
mean(B.samples[31:60,i])) #simulate difference in means to compare to
test.stat
      }
      p.values[count] <- mean (Boot.test.stat >= test.stat) #record p-
value, assuming H0: no difference in city-algo.

      cat(name1, "," , name2 , "," , name3 , "," , name4 , "i=" , count,
"," , "obs. diff. in mean=" , test.stat, "\n")
      count <- count + 1
    }
  }
}
print(p.values)
p.adj <- p.adjust(p.values, method = "bonferroni", n = length(p.values))
print(p.adj) #adjust p-values to control for Family-wise error rate / Type
I error
alpha <- 0.05
which(p.adj > alpha) #88 91
###88: Palm Coast-Neighbors vs. Austin-RF with adj. p-value = 0.105
###91: Austin-NN vs. Mesa-NN with adj. p-value = 0.5355
###are all NOT statistically significant, the rest are significant.

#Precision
n <- 30*2
B <- 10000
Cities <- c("Austin","Mesa","Palm_Coast")
Algorithms <-
c("Bayes","Catboost","Neighbors","Neural_Network","Random_Forest")
veclist <- do.call(paste,expand.grid(Cities,Algorithms))
count <- 1
p.values <- rep(-1,105)
for(i in 1:15){
  temp <- strsplit(veclist[i], " ")[[1]]
  name1 <- gsub("_", " ", temp[1]) #City
  name2 <- gsub("_", " ", temp[2]) #Algo
  for(j in (i+1):15){
    temp2 <- strsplit(veclist[j], " ")[[1]]
    name3 <- gsub("_", " ", temp2[1]) #City
    name4 <- gsub("_", " ", temp2[2]) #Algo
    if( i != 15){ #prevent last two comparisons, so theres ((12-1)*12) / 2
= 66 total comparisons
      variable1 <- precision.df$Precision[precision.df$City == name1 &
precision.df$Algorithm == name2]
      variable2 <- precision.df$Precision[precision.df$City == name3 &
precision.df$Algorithm == name4]
      variable <- c(variable1, variable2)
      #Analysis
      test.stat <- abs(mean(variable1) - mean(variable2))
      B.samples <- matrix(sample(variable, size = n*B, replace=TRUE),
                          nrow=n, ncol=B)
      Boot.test.stat <- rep(0,B)
      for(i in 1:B){
        Boot.test.stat[i] <- abs(mean(B.samples[1:30,i]) -
mean(B.samples[31:60,i]))
      }
      p.values[count] <- mean (Boot.test.stat >= test.stat)

      cat(name1, "," , name2 , "," , name3 , "," , name4 , "i=" , count,
"," , "obs. diff. in mean=" , test.stat, "\n")
      count <- count + 1
    }
  }
}
print(p.values)
p.adj <- p.adjust(p.values, method = "bonferroni", n = length(p.values))
print(p.adj) #adjust p-values to control for Family-wise error rate / Type
I error
alpha <- 0.05
which(p.adj > alpha) #78, 84, 91, 94, 98
###78: Mesa-Neighbors vs. Palm Coast-Neighbors with adj. p-value = 1.00
###84: Mesa-Neighbors vs. Palm Coast-RF with adj. p-value = 0.063
###91: Austin-NN vs. Mesa-NN with adj. p-value = 0.357
###94: Austin-NN vs. Mesa-RF with adj. p-value = 1.00
###98: Mesa-NN vs. Mesa-RF with adj. p-value = 0.1365
###are all NOT statistically significant, the rest are significant.

#Recall
n <- 30*2
B <- 10000
Cities <- c("Austin","Mesa","Palm_Coast")
Algorithms <-
c("Bayes","Catboost","Neighbors","Neural_Network","Random_Forest")
veclist <- do.call(paste,expand.grid(Cities,Algorithms))
count <- 1
p.values <- rep(-1,105)
for(i in 1:15){
  temp <- strsplit(veclist[i], " ")[[1]]
  name1 <- gsub("_", " ", temp[1]) #City
  name2 <- gsub("_", " ", temp[2]) #Algo
  for(j in (i+1):15){
    temp2 <- strsplit(veclist[j], " ")[[1]]
    name3 <- gsub("_", " ", temp2[1]) #City
    name4 <- gsub("_", " ", temp2[2]) #Algo
    if( i != 15){ #prevent last two comparisons, so theres ((12-1)*12) / 2
= 66 total comparisons
      variable1 <- recall.df$Recall[recall.df$City == name1 &
recall.df$Algorithm == name2]
      variable2 <- recall.df$Recall[recall.df$City == name3 &
recall.df$Algorithm == name4]
      variable <- c(variable1, variable2)
      #Analysis
      test.stat <- abs(mean(variable1) - mean(variable2))
      B.samples <- matrix(sample(variable, size = n*B, replace=TRUE),
                          nrow=n, ncol=B)
      Boot.test.stat <- rep(0,B)
      for(i in 1:B){
        Boot.test.stat[i] <- abs(mean(B.samples[1:30,i]) -
mean(B.samples[31:60,i]))
      }
      p.values[count] <- mean (Boot.test.stat >= test.stat)

      cat(name1, "," , name2 , "," , name3 , "," , name4 , "i=" , count,
"," , "obs. diff. in mean=" , test.stat, "\n")
      count <- count + 1
    }
  }
}
print(p.values)
p.adj <- p.adjust(p.values, method = "bonferroni", n = length(p.values))
print(p.adj) #adjust p-values to control for Family-wise error rate / Type
I error
alpha <- 0.05
which(p.adj > alpha) #48, 78, 96
###48: Austin-Catboost vs. Austin-RF with adj. p-value = 1.00
###78: Mesa-Neighbors vs. Palm Coast-Neighbors with adj. p-value = 1.00
###96: Mesa-NN vs. Palm Coast-NN with adj. p-value = 1.00
###are all NOT statistically significant, the rest are significant.

####FAILED METHODS BELOW###########################################
#LOG TRANSOFRMATIONS DIDN'T WORK

#new models
#data visualization
ggplot(accuracy.df, aes(log(Accuracy), Algorithm, colour = City)) +
geom_point()
ggplot(precision.df, aes(log(Precision), Algorithm, colour = City)) +
geom_point()
ggplot(recall.df, aes(log(Recall), Algorithm, colour = City)) +
geom_point()
#models
```

```
accuracy.model <- aov(log(Accuracy) ~ City*Algorithm, data = accuracy.df)
precision.model <- aov(log(Precision) ~ City*Algorithm, data =
precision.df)
recall.model <- aov(log(Recall) ~ City*Algorithm, data = recall.df)
#analysis of overall model
summary(accuracy.model)
summary(precision.model)
summary(recall.model)
#residual plots and assumptions
#1. Homogoneity of variance - Violated
#Because all n_i = 30, ANOVA ca be robust to small/medium violations
plot(accuracy.model,1)
plot(precision.model,1)
plot(recall.model,1)
leveneTest(log(Accuracy) ~ City*Algorithm, data = accuracy.df)
leveneTest(log(Precision) ~ City*Algorithm, data = precision.df)
leveneTest(log(Recall) ~ City*Algorithm, data = recall.df)
#2. Normality - Violated
plot(accuracy.model,2)
plot(precision.model,2)
plot(recall.model,2)
acc.res <- residuals(object = accuracy.model)
pre.res <- residuals(object = precision.model)
rec.res <- residuals(object = recall.model)
shapiro.test(x = acc.res)
shapiro.test(x = pre.res)
shapiro.test(x = rec.res)

#TRANSFORMATIONS DIDN'T WORK
#finding lambda
library(MASS)
out <-
boxcox(lm(accuracy.df$Accuracy~accuracy.df$City*accuracy.df$Algorithm))
range(out$x[out$y > max(out$y)-qchisq(0.95,1)/2])
(-1.3535354 + 0.5454545) / 2

out.p <-
boxcox(lm(precision.df$Precision~precision.df$City*precision.df$Algorithm)
)
range(out.p$x[out.p$y > max(out.p$y)-qchisq(0.95,1)/2])
(-2.000000 + -1.515152) / 2

out.r <- boxcox(lm(recall.df$Recall~recall.df$City*recall.df$Algorithm))
range(out.r$x[out.r$y > max(out.r$y)-qchisq(0.95,1)/2])
(-1.919192 + 1.070707) / 2
#new models
#data visualization
ggplot(accuracy.df, aes(Accuracy^(-1/2), Algorithm, colour = City)) +
geom_point()
ggplot(precision.df, aes(Precision^(-1.75), Algorithm, colour = City)) +
geom_point()
ggplot(recall.df, aes(Recall^(-1/2), Algorithm, colour = City)) +
geom_point()
#models
```

```
accuracy.model <- aov(Accuracy^(-1/2) ~ City*Algorithm, data =
accuracy.df)
precision.model <- aov(Precision^(-1.75) ~ City*Algorithm, data =
precision.df)
recall.model <- aov(Recall^(-1/2) ~ City*Algorithm, data = recall.df)
#analysis of overall model
summary(accuracy.model)
summary(precision.model)
summary(recall.model)
#residual plots and assumptions
#1. Homogoneity of variance - Violated
#Because all n_i = 30, ANOVA ca be robust to small/medium violations
plot(accuracy.model,1)
plot(precision.model,1)
plot(recall.model,1)
leveneTest(Accuracy^(-1/2) ~ City*Algorithm, data = accuracy.df)
leveneTest(Precision^(-1.75) ~ City*Algorithm, data = precision.df)
leveneTest(Recall^(-1/2) ~ City*Algorithm, data = recall.df)
#2. Normality - Violated
plot(accuracy.model,2)
plot(precision.model,2)
plot(recall.model,2)
acc.res <- residuals(object = accuracy.model)
pre.res <- residuals(object = precision.model)
rec.res <- residuals(object = recall.model)
shapiro.test(x = acc.res)
shapiro.test(x = pre.res)
shapiro.test(x = rec.res)
```