Summer 2016

Detecting Fires: A Nationally Consistent, Rule Based Approach

Jacob D. (Jacob Daniel) Lesser
Western Washington University, lesserj@wwu.edu

Detecting Fires:
A Nationally Consistent, Rule Based Approach

By

Jacob Daniel Lesser

Accepted in Partial Completion
Of the Requirements for the Degree
Master of Science

Kathleen L. Kitto, Dean of the Graduate School

ADVISORY COMMITTEE

Chair, Dr. Michael J. Medler

Dr. David O. Wallin

Dr. Douglas F. Smith

MASTER'S THESIS

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Western Washington University, I grant to Western Washington University the non-exclusive royalty-free right to archive, reproduce, distribute, and display the thesis in any and all forms, including electronic format, via any digital library mechanisms maintained by WWU.

I represent and warrant this is my original work, and does not infringe or violate any rights of others. I warrant that I have obtained written permissions from the owner of any third party copyrighted material included in these files.

I acknowledge that I retain ownership rights to the copyright of this work, including but not limited to the right to use all or part of this work in future works, such as articles or books.

Library users are granted permission for individual, research and non-commercial reproduction of this work for educational purposes only. Any further digital posting of this document requires specific permission from the author.

Any copying or publication of this thesis for commercial purposes, or for financial gain, is not allowed without my written permission.

Jacob D. Lesser
July 29, 2016

Detecting Fires:
A Nationally Consistent, Rule Based Approach

A Thesis
Presented to
The Faculty of
Western Washington University

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science

By
Jacob Daniel Lesser
June 2016

Abstract

One of the continuing challenges in wildland fire management is maintaining accurate vegetation and fuel data of an adequate resolution on an ever-changing landscape. The USGS's LANDFIRE program produces national, mid-level resolution datasets of fuel, vegetation, and fire regime data useful in the modeling of wildland fire behavior. One of the most effective and least expensive ways for maintaining the accuracy of these layers is to incorporate area updates by detecting landscape changes. While many algorithms exist for detecting change and disturbances, these algorithms are often tuned for a particular landscape and require very precise training data or rely heavily on scene statistics. This research looks at a method for detecting wildland fire across a broad array of landscapes using a collection of computer-generated rules built from hundreds of thousands of points of training data. Verification of the results were assessed by visual comparison to a time series of high spatial resolution imagery through Google Earth and cross-referenced to fires from various historical databases. A majority of the fires detected in this assessment were in either a conifer or grassland landscape. The methods outlined in this thesis performed best in those two landscapes, detecting 73% to 78% of conifer fires correctly and 79% to 83% of grassland fires correctly.

Acknowledgements

I would like to thank my thesis committee, Michael Medler, David Wallin and Gus Smith for their enduring patience, support and advice on this long journey. I would like to thank the fine people in the USGS's Landfire program: Matt Rollins, Dan Steinwand, Mike Coan, Jeff Eidenshink, Don Ohlen, Josh Picotte, Stephen Howard and Kurtis Nelson. This research was a product of their vision and dedication to the creation of a high quality LANDFIRE dataset. I would also like to thank my family and friends for supporting me in this process and helping me stay focused. To my fellow grad students: This was a shared experience and one which would never have been the same without your comradery. Thank you for the experience and the friendship. Finally, a special thanks to WWU's Cross Country and Track and Field teams. You provided me an outlet and helped me stay grounded. You all have my enduring thanks!

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BAER	Burned Area Emergency Response
BARC	Burned Area Reflectance Classification
CV	Change Vector
dNBR	Differenced Normalized Burn Ratio
dNDVI	Differenced Normalized Difference Vegetation Index
EROS	Earth Resources Observation and Science Center
FOD	Fire Occurrence Database
Gb	Gigabyte
GDAL	Geospatial Data Abstraction Library
GIS	Geographic Information System
LANDFIRE	Landscape Fire and Resource Management Planning Tools
LEDAPS	Landsat Ecosystem Disturbance Adaptive Processing System
Mb	Megabyte
MIICA	Multi-Index Integrated Change Analysis
MODIS	Moderate Resolution Imaging Spectroradiometer
MRLC	Multi-Resolution Land Characteristics
MTBS	Monitoring Trends in Burn Severity
NBR	Normalized Burn Ratio
NDVI	Normalized Difference Vegetation Index
NIR	Near-Infrared
NLCD	National Land Cover Database
RCVMAX	Relative Change Vector Maximum
RGB	Red, Green, Blue
SWIR	Shortwave-Infrared
USGS	United States Geological Survey
UTM	Universal Transverse Mercator
VCT	Vegetation Change Tracker
WRS Path/Row	Worldwide Reference System Path/Row

Introduction

The modern computer, running on silicon-based processors, was invented in the early 1970's. Since then, the computational power of a single computer has doubled nearly every eighteen months[1]. The rapid expansion of low cost computing resources has made the storage of large datasets and the analysis of data using computationally intensive algorithms possible on inexpensive consumer hardware. In this research thesis, I took advantage of this rapid growth in computing resources to develop a process for detecting disturbances using a time series of Landsat imagery and a machine-learning algorithm.

In this research, I extracted hundreds of thousands of points of spectral response data from a three year, six image time series of Landsat satellite images. I extracted the spectral data from Landsat in locations and time periods where wildland fire had occurred within the past year. I also extracted data from areas where wildland fire had not recently occurred. This data was combined into a training dataset and processed with a machine-learning algorithm called Cubist (Quinlan, 2016) that employs a

[1] According to Moore's Law, it is the number of transistors on an integrated circuit that doubles every eighteen months. However, the number of transistors on a chip and the chips computational power are not strictly a 1:1 relationship. As you add more transistors to a chip, you have greater power consumption and require greater heat dissipation. The higher heat associated with chips with more transistors can have a negative impact on processing capabilities. However, generally speaking, the number of transistors strongly correlates with processing power, or the number of calculations that can be performed per second.

multivariate linear model algorithm. I used Cubist to analyze this dataset for temporal patterns in spectral trajectories. Cubist turned those patterns into a series of rules. Each rule described a multivariate linear model. A multivariate linear model is similar to a linear regression model, but has some key differences. Primarily, multivariate linear models have a linear model at each leaf, while a regression model has a simple numeric output (Walsh & Milligan, 2005; Walton, 2008). The value of each equation, when applied to each pixel in a new Landsat time series, can be used to predict the likelihood of whether or not a fire had occurred in that location within the past year.

One major advantage of machine-learning algorithms, like Cubist, are their ability to identify patterns within large and noisy datasets. These machine-learning algorithms allow researchers to analyze "fuzzy" datasets where a percentage of the input data are labeled incorrectly. Outliers in the data can be detected and ignored by Cubist when building rules that describe the data. This allows researchers to use many of the disturbance datasets available despite the variability in data quality and reliability. These algorithms can therefore accept a percentage of incorrectly labeled data. i.e. training data labeled as a wildland fire that was not actually a fire or data that was not labeled as fire, even when it should have been. This is very important in change detection models, because much of the available data are of varying quality.

Keeping large GIS databases such as the USGS's LANDFIRE database up to date are major undertakings. LANDFIRE is a joint program of the Department of Agriculture and the Department of the Interior. LANDFIRE provides 30m pixel

resolution datasets for the whole US. LANDFIRE datasets include existing and potential vegetation layers, surface and canopy fuel layers important for modeling fire behavior, fire regime datasets that describe historical fire characteristics, and topographic layers. The LANDFIRE program currently employs several analysts that look at disturbance GIS data from a variety of federal, state and local sources as well as disturbances detected with automated computer algorithms. LANDFIRE analysts overlay these disturbance datasets on Landsat and other high-resolution imagery data using Hexagon Geospatial's ERDAS Imagine and ESRI's ArcGIS computer software programs. When those analysts manually confirm the location and type of disturbance through visual interpretation of the data, they integrate the changes associated with that disturbance back into the appropriate LANDFIRE layers (B. K. J. Nelson et al., 2016). For example, when a fire burns through an area and removes dead and woody debris, the analyst updates the fuel models in that area. When agricultural land is replaced with a housing suburb, the analyst updates the underlying vegetation layer to reflect those changes.

Much of the change information used today comes from Landsat and Landsat-derived products. Landsat 7 currently transmits approximately 400 images a day to Earth for processing while Landsat 8 transmits upwards of 700 images per day (U.S. Geological Survey, 2015). Each image covers an area approximately 31,000 square kilometers. Finding and identifying all changes by hand would require an inordinate amount of time and staff. Instead, the LANDFIRE analysts currently rely on automated

change detection and disturbance detection modeling routines. When a disturbance is detected by a computer algorithm, the analyst inspects that disturbance and attempts to associate it with a known disturbance event (K. J. Nelson, Connot, Peterson, & Martin, 2013). If the detected disturbance cannot be associated with a known disturbance event, the LANDFIRE analyst attempts to determine disturbance type (K. J. Nelson et al., 2013). The disturbance can then be delineated and integrated back into the appropriate LANDFIRE layers by the analyst. Currently, the USGS's LANDFIRE program uses the Multi-Index Integrated Change Analysis (MIICA) change detection algorithm, recently replacing the Vegetation Change Tracker (VCT) algorithm, to perform LANDFIRE's automated change detection routines (B. K. J. Nelson et al., 2016).

Objective

The objective of this research thesis was to develop a tool using the machine-learning algorithm Cubist that could supplement or replace the current MIICA algorithm by providing more complete wildland fire detections. This tool will improve the LANDFIRE analysts' workflow by giving them the tools to quickly identify and verify wildland fire disturbances while detecting fires not currently detected by existing algorithms. Increased detection performance and better detection verification workflows will help make LANDFIRE a better product.

Unlike the current MIICA method, this machine-learning technique can integrate the analysts' verification data back into the training dataset, improving Cubist's rule model and increasing the efficacy of the algorithm over subsequent iterations.

This thesis focuses specifically on the detections of wildland fire as a test case for using the Cubist algorithm as a detection method for multiple disturbance types. In order to aid the analysts in quickly verifying wildland fire detections, I built a web based tool for looking at the detections and comparing against available ancillary data and high-resolution imagery. This allows the analyst to quickly identify and flag false detections while keeping the correct detections for further analysis. As false detections are identified, these cases can be integrated back into the training dataset, potentially improving future iterations of Cubist's rule model.

Background

Why Change Detection is Important

An understanding of landscape composition (e.g. topology, vegetation, and fuel dynamics) is critical for making well-informed land management decisions. Accurate data about the landscape can give policymakers the tools to make better management decisions about the land within their jurisdiction (Dale et al., 2000). High quality vegetation, fuel, and topographic datasets also give wildland fire managers' better ability to model and predict fire behavior, giving them the tools to best allocate resources and protect the lives of the firefighters under their charge (Rollins, 2009).

Mapping all the vegetation cover for the United States is an expensive and time consuming endeavor (Rollins, 2009). The United States is nearly one billion hectares in size. Of that, 259 million hectares, or about 28% of all land in the United States are

publicly owned, (Vincent, Hanson, & Bjelopera, 2014). It is not reasonable, nor feasible to map all public and/or private land every year. However, the landscape is constantly changing. Wildland fire burns fuel and resets successional stages of the landscape (Wright, 1974). Insects damage forests making them more susceptible to fire and disease (Parker, Clancy, & Mathiasen, 2006). Forests in the US are thinned and cleared at the rate of 4.4 million hectares annually (Masek et al., 2011). New trees are planted at the rate of 900,000 hectares annually (Masek et al., 2011). Farmlands are developed into shopping malls and homes. New roads crisscross the landscape, allowing an ever growing population to move into and develop more remote areas (Theobald & Romme, 2007; Tully, 2013). Keeping track of all this change is difficult, but important for keeping data layers, such as LANDFIRE, up to date. While it is not feasible to remap the landscape every year with current technology, personnel, and budgets, identifying the changes on the landscape using big data analytics and machine-learning techniques, and making inferences on how those changes affect the landscape, will allow data managers to keep those critical data layers up to date.

History of Airborne and Spaceborne Imagery

People have been putting cameras into the air and pointing them at the ground shortly after the camera's invention (J Hannavy, 2013). As early as the 1880's, cameras were attached to kites, balloons, and even the occasional pigeon (Cohen & Goward, 2004; J Hannavy, 2013; Hildebrandt, 1908). In these early days remotely sensed images were typically one-band black and white photos (Olsen, 2007). Over time, as cameras, film, and sensors became more sophisticated, we were able to capture more data at higher resolutions and with color and infrared bands of information. However, until late in the last century, most of these images were analyzed by hand by specially trained analysts (Balchin, 1987). This did not scale well and it limited our ability to analyze the ever growing amount of data. Mapping was one of the first objectives of remote sensing, but soon, the potential of aerial imagery for military applications were realized, driven in large part by the first two world wars (Lillesand, Kiefer, & Chipman, 2015). The first 100 years of airborne and spaceborne imaging applications were dominated by military and state spy craft objectives with images coming from either spy-planes or military satellites (Downing, 2011). It wasn't until 1972, and the birth of the environmental movement, that the first Land Remote-Sensing Satellite (Landsat) was launched into low-earth orbit (Cohen & Goward, 2004). Landsat 1 was equipped with a 4 band multispectral camera tuned for environmental monitoring (Table 1), starting an era of remote sensing for environmental monitoring

Figure 1 – Pigeon Camera from 1914. Attribution: Bundesarchiv, Bild 183-R01996 / CC-BY-SA 3.0

and landscape analysis (Cohen & Goward, 2004). With the launch of Landsat, researchers were now able to use remotely gathered satellite images to begin analyzing the natural features of the Earth. Much of this analysis in the early days continued to be done by hand or with simple supervised and unsupervised classification techniques, but with increased access to modern computers, researchers began developing algorithms to detect features and classify landscapes automatically (Fleming, Berkebile, & Hoffer, 1975; Mirajkar & Srinjvasan, n.d.). As computers and algorithms became more sophisticated, more complex problems could be applied to a series of images to detect temporal patterns of change, such as landscape disturbance. Initially these algorithms were relatively simple, using a handful of carefully collected, collated, and hand-tuned training points to describe all the features the analyst hoped to detect (Fleming et al., 1975). This process of collecting training points was time consuming and expensive as it often required field checking the training data points. It also assumed the analysts' biases in picking training points that they thought best described the phenomena they were trying to identify. Once these training points were collected, a series of classified images could then be compared and differences between the two analyzed.

Table 1 – Landsat band wavelengths. The first three Landsat satellites used the Multispectral scanner and consisted of four spectral bands of information at approximately 60m spatial resolution. Landsat 4 and 5 used the Thematic Mapper scanner. All but the thermal band are recorded at a 30m spatial resolution. The thermal band (6) is recorded at 120m spatial resolution. Landsat 7 uses the Enhanced Thematic Mapper Plus scanner. The spectral band information recorded by Landsat 7 mirrors that of 4 and 5. However, Landsat 7 adds an additional panchromatic band (8) that records information at 15m pixel resolution. Landsat 8 uses the Operational Land Imager and adjusts the spectral band information recorded from that of its predecessors. Additionally, Landsat 8 adds a band (9) used for detecting cirrus clouds. Landsat 8 also includes two new thermal bands on a different sensor not included in this table (U.S. Geological Survey, 2016). *Landsat 6 carried the ETM sensor, but failed to achieve orbit.*

	Landsat 1-3 (MSS)	*Landsat 4-5 (TM)*	*Landsat 7 (ETM+)*	*Landsat 8 (OLI)*
Band 1	***N/A***	***0.45-0.52***	***0.45-0.52***	***0.43 - 0.45***
Band 2	***N/A***	***0.52-0.60***	***0.52-0.60***	***0.45 - 0.51***
Band 3	***N/A***	***0.63-0.69***	***0.63-0.69***	***0.53 - 0.59***
Band 4	***0.5-0.6***	***0.76-0.90***	***0.77-0.90***	***0.64 - 0.67***
Band 5	***0.6-0.7***	***1.55-1.75***	***1.55-1.75***	***0.85 - 0.88***
Band 6	***0.7-0.8***	***10.40-12.50***	***10.40-12.50***	***1.57 - 1.65***
Band 7	***0.8-1.1***	***2.08-2.35***	***2.09-2.35***	***2.11 - 2.29***
Band 8	***N/A***	***N/A***	***.52-.90***	***0.50 - 0.68***
Band 9	***N/A***	***N/A***	***N/A***	***1.36 - 1.38***

In the decades since, more Landsat satellites have been launched, each making improvements and refinements over previous models. The latest version of Landsat, Landsat 8, images the earth recording 7 bands of data at 30-meter spatial resolution, plus two thermal bands at lower spatial resolution. In the versions between Landsat 1 and 8, the number of bands have increased and the wavelength intervals for each band have been refined to optimize for environmental landscape analysis (Table 1) (U.S. Geological Survey, 2016). This thesis primarily relies on Landsat 5 data because of its long-term data availability and reliability.

Changes in the Computer Environment

The early 1970's marked the beginning of the silicon-based processor era. In 1972, the top of the line Intel 8008 processor had 3,500 transistors, could access 16kb of memory and perform 60,000 instructions per second (Betker, Fernando, & Whalen,

1997). One 8008 processor cost over $650 in 2016 dollars (Betker et al., 1997). Since the 1970's however, processing power has followed Moore's law, (until very recently) doubling every 18 months, while the cost per transistor has decreased by half every 1.1 years (Kurzweil, 2016)[2]. A midrange consumer-level Intel processor in 2011, the latest for which Intel published numbers, had 1.16 billion transistors and cost about $300 (McGrath, 2011).

Prior to the 1980's, most data storage was on magnetic tape. A high-end typical tape storage disk in the mid 1970's could store between 5Mb and 140Mb of data. A typical Landsat 5 scene for all 7 bands is approximately 1.5Gb in size[3]. The limitations in tape capacity meant computer storage and digital analysis of Landsat and other aerial imagery was difficult, time consuming, and prohibitively expensive outside of large research organizations and Universities. It wasn't until the 1980's that magnetic storage hard drives became available, and early on, their capacity was tiny and their cost high. The first consumer hard disk drives were released in 1980 with a total capacity of 26Mb. The cost per Gb in hard drive storage in 1980 was $437,500, while in

[2] Gordon Moore was the co-founder of Intel. In 1965, he predicted that the number of components on an integrated circuit would double every year. He later refined that prediction to every two years. His prediction, on average, held true until 2012. The typical transistor size in 2012 was as small as 22nm and is approaching 10nm in 2016. The size of silicon based transistors are reaching their physical limit of about 5nm. Any smaller than that, and the transistor starts to experience the effects of quantum mechanics. However, researchers have been experimenting with new materials that will allow even smaller transistors. Transistors made out of newer materials, such as graphene, have been produced in a lab as small as 10 atoms think. As the end of the silicon era is approaching and new materials come into use, Moore's law will likely need to be revised.

[3] 1Gb is equal to 1024Mb. A typical Landsat image would require anywhere from 10 to 300 tapes.

2013, the cost per Gb was as low as 4.3 cents (Statistic Brain Research Institute, 2016). Performing remote sensing analysis in the early days was extremely costly and slow (Cohen & Goward, 2004).

Many of the machine-learning algorithms we use today are simply too resource intensive to even be considered on the most sophisticated hardware of the early era of environmental remote sensing. The rapid increase in computing power has allowed us to build much more sophisticated, computational and data intensive algorithms. This has given rise to the machine-learning era where we can feed large quantities of data into the computer and let the computer, using sophisticated algorithms, identify patterns that make up the processes we are trying to identify.

Remote sensing techniques have evolved over time and have gotten more sophisticated as computing capabilities have increased. The quantity of available data is ever growing. As of 2011, the growth rate of stored data worldwide was growing 25% annually at a compounding rate (Press, 2013). This rapid rate of data growth poses many challenges in mining the large quantities of data for useful information and patterns. However, as these big data tools evolve, a natural application for those tools will be in the remote sensing arena.

Remote Sensing Techniques

Computer driven remote sensing originally relied on analysis of a single image analysis using either supervised or unsupervised classification techniques (Lillestrand, 1972). Supervised classification relies on labeled training data to inform the spectral

signatures of the desired targets of discovery. Unsupervised classification breaks the imagery up into areas of similar spectral signatures and leaves it up to human interpretation as to what those areas represent. Overtime, the algorithms for both supervised and unsupervised classification techniques have improved. Eventually these techniques were expanded to multi-imagery comparisons looking for change between imagery taken on two or more different dates (Singh, 1989). The simplest analysis would be to subtract one band of data in one scene from the same band of data from another scene from two spatially equivalent images. The difference of these scenes would inform a potential change in landscape. This simple analysis, however, doesn't inform the type of change, and doesn't account for seasonal differences between images. More sophisticated techniques looking for patterns between multi-bands of data from images on multiple dates have evolved to identify changes and trends and allows the algorithm to identify only specified phenomena, if so desired.

The first several versions of the LANDFIRE data products relied on the Vegetation Change Tracker (VCT) algorithm to detect change and inform updates to the LANDFIRE dataset. VCT is algorithmically similar to Oregon State University's LandTrendr project (Kennedy, Yang, & Cohen, 2010), both being developed in 2008, shortly after the Landsat archive became freely available. In the latest 1.3.0 version of LANDFIRE, VCT was replaced by the Multi-Index Integrated Change Analysis (MIICA) algorithm as the primary source of change detection data (B. K. J. Nelson et al., 2016).

Disturbance Detection with LandTrendr

LandTrendr is a change detection tool developed at Oregon State University in cooperation with the U.S. Forest Service and NASA. Unlike many traditional change detection routines that only look for change in a pair of scenes, LandTrendr looks for change by analyzing the trends of individual pixels over the course of many years. Not only does this allow the algorithm to pick up abrupt disturbances on the landscape such as fire and clear-cuts, but also long-term trends of vegetation decline and regrowth (Kennedy et al., 2010).

Trends are identified by stacking pre-processed Landsat scenes taken in the summer months from multiple years and graphing the spectral values of a single band or index (based on two or more bands) for those years. Best fit vertices are created through a series of mathematical functions that simplify the vertices into generalized trends (Kennedy et al., 2010). This is repeated for every pixel in a scene.

These trends are then compared to an idealized trend model for varying types of change. If the trend shapes are similar enough within a statistical margin-of-error, then the trend is labeled as a change. The model includes four idealized trend patterns: simple disturbance, disturbance followed by revegetation, ongoing revegetation, and revegetation to stable state. Everything else is labeled as not having changed. In addition to labeling change, the year of disturbance, its intensity and the rate of recovery were also labeled. While the LandTrendr model can automatically detect disturbance and change directionality, it does not classify the cause of change. A

separate process where groups of pixels have been identified as changed are validated and labeled by a human analyst comparing the LandTrendr output and high-resolution satellite photos (Cohen, Yang, & Kennedy, 2010; Kennedy et al., 2010).

Disturbance Detection with Vegetation Change Tracker (VCT)

The Vegetation Change Tracker (VCT) was developed as part of the USGS's LANDFIRE program located at the EROS Data Center in Sioux Falls, South Dakota (Vogelmann et al., 2011). It is algorithmically similar to that of LandTrendr. VCT is able to accurately detect most stand-clearing disturbances including fire and clear-cuts and many other disturbances that don't have complete stand removal (K. J. Nelson et al., 2013). The VCT algorithm is tuned specifically to forested environments but tends to break down in non-forest environments (Jin et al., 2013). This is in part dependent on VCT's dependence on scene statistics and the brightness of many non-forested scenes prevent the bright spots of fire from being detected (Jin et al., 2013).

Disturbance Detection with MIICA

In 2013 the LANDFIRE program switched its change detection algorithm to the Multi-Index Integrated Change Analysis (MIICA) algorithm (B. K. J. Nelson et al., 2016). This algorithm was developed by the Multi-Resolution Land Characteristics (MRLC) consortium. MRLC produces the National Land Cover Database (NLCD) and developed MIICA for the 2011 NLCD update, which was the last major NLCD update. MIICA uses four spectral indices derived from a pair of Landsat scenes to find changes

between two dates in time. This algorithm performs better at detecting disturbances, like wildland fire, in non-forested areas than the VCT algorithm.

The indices used in MIICA are the differenced Normalized Burn Ratio (dNBR), the differenced Normalized Difference Vegetation Index (dNDVI), the Change Vector (CV), and an index developed specifically for MIICA called the Relative Change Vector Maximum (RCVMAX) index (Jin et al., 2013).

Detecting Fire

My research will focus on the development of an alternative approach for the detection of wildland fire events. This approach may also be useful for detecting other disturbance types. My research is facilitated by the availability of the US Forest Services Fire Occurrence Database (FOD) of fire points for the years 1992-2013 (Karen C. Short, 2015). This is the most complete database of fire in the US. While this database is certainly not a complete record of wildland fires in the United States, it is far more complete than any other previous database (K. C. Short, 2014). To the best of my knowledge, no similar databases of nationally complete data for both public and private land exists for other disturbances of interest, such as clear-cuts, thinning, insect damage, or other disturbances. The availability of this database was useful in verifying the detection results.

There are many methods of fire detection and they serve different purposes. Wildland fire can have significant impact on the human environment, so having rapid, near real-time detection methods for fires is important for identifying active fires that

might require immediate human response. The primary, global, real-time detection method for wildland fires relies on the Moderate Resolution Imaging Spectroradiometer, more commonly referred to as MODIS, satellite data.

MODIS is an imaging platform aboard the Terra and Aqua EOS Satellites. Both satellites are in near-polar orbits and are both sun-synchronous. The spatial resolution and orbital paths of the two MODIS platform satellites enable the earth to be imaged four times a day – twice at night and twice in the day. The high-temporal frequency of global passes allows for detection of near real-time events and provides early warning and analysis of a disturbance (Giglio, Csiszar, & Justice, 2006).

The MODIS fire detection system is an early-warning tool for detecting the spatial location of fires across the planet. Fires can be detected at the sub-pixel level (Giglio et al., 2006). However, given the 1 kilometer pixel resolution, it is not a good tool for the assessment of fire perimeters or internal heterogeneity of fires. It is also not a tool for detecting fire scars on the landscape. It is solely a real-time active fire detection tool.

Other algorithms, such as VCT, LandTrendr, and the research developed in this thesis, look for past fires based on a time series of Landsat imagery by looking for changes in the recorded spectral data that is indicative of a fire scar.

Detecting Disturbance vs. Delineation of Disturbance Area

When looking for wildland fires and other disturbances with remote sensing techniques, there are three levels of detail possible. The first level is the detection of the disturbance. Detection of a disturbance will find the location of a change, but not provide details about size and severity. The second level is delineation of the disturbance perimeter. Delineation of the perimeter will provide details on size and location of the disturbance, but will not provide detail about internal severity of individual pixels within the disturbance. Finally, delineation with classification will determine the location, spatial extent and internal heterogeneity of the disturbance by classifying the level of disturbance of each individual pixel within the disturbance perimeter.

While delineation and classification of disturbances is an important future goal for this research, this thesis focuses on detection of wildland fire. This is in large part due to the lack of complete historical fire records, including location, perimeter, and severity of fires. This makes verification of fire perimeters and internal heterogeneity difficult. The FOD database, which is the best fire occurrence database to date, only includes point data, so shape and internal heterogeneity of the detected fires can't be compared to the outputs of my model. While delineation would be beneficial for the advancement of wildland fire research, positive identification of previously unknown fires is the first goal of this research. Detection of previously undetected fires will allow

the LANDFIRE analysts to manually delineate the disturbance and further improve the LANDFIRE data products.

Utility of Research

Identifying wildland fires and other disturbances is important for a number of reasons. It helps data managers, such as the USGS's LANDFIRE team, keep fuel and vegetation layers up to date by incorporating the changes associated with disturbances into existing datasets. For example, this could include updates to fuel load and classification layers as fuel is removed by fire. Incorporating the results of these changes into the data layers can improve the accuracy and decrease the cost of maintaining those datasets by eliminating the need to remap the data in its entirety. Knowing the location and size of fires can help policy makers and researchers better understand the wildland fire landscape. Good fire data, for example, can help determine trends in the number of ignitions, total fire area, and severity of fire. These are important pieces of information when studying the trends and impacts of climate change. Having accurate fuel, vegetation and historical fire data is important for informing wildland fire policy and modeling fire behavior. The goal of this thesis is to develop a tool for quickly and accurately identifying past wildland fires. More complete and accurate data can be used by LANDFIRE analysts to improve the LANDFIRE data products.

Machine-Learning and Cubist

Machine-learning is the ability for computers to identify patterns without explicitly being programmed to recognize a predetermined pattern (Samuel, 1959). Machine-learning algorithms identify patterns in data and create models that predict outcomes based on that pattern given similar data. These algorithms can continually improve their models as more data is collected, classified, and input into the algorithm. Generally speaking, machine-learning algorithms are broken down into two broad types: supervised learning and unsupervised learning (Russell & Norvig, 2009). Unsupervised learning algorithms analyze a dataset for structure without knowing anything about the meaning of the dataset. Supervised learning is a technique where data are labeled within a dataset by type and the data are analyzed for patterns describing each of the labeled types.

There are many types of machine-learning algorithms, all with different strengths and weaknesses. A non-exclusive list of machine-learning algorithms includes decision trees, regression trees, neural networks, deep learning, Bayesian networks, and genetic algorithms. For this thesis, I am using a machine-learning algorithm called Cubist which is a multivariate linear model algorithm (Quinlan, 2016).

Cubist is an algorithm that builds a collection of rules based on the patterns found within the data. Each rule describes a multivariate linear model. That linear model can then be applied to each pixel value in a Landsat pixel stack that calculates a numerical value. This numerical value describes the likelihood that a particular value is

correctly identified. The analyst can set a threshold to convert these numerical values into a binary response. In this case, I set a threshold of 95. Values above 95 are considered burned, while values below 95 are considered unburned. Cubist, according to the documentation "… is a powerful tool for generating rule-based models that balance the need for accurate prediction against the requirements of intelligibility. Cubist models generally give better results than those produced by simple techniques such as multivariate linear regression, while also being easier to understand than neural networks" (Quinlan, 2016). Cubist is designed to quickly analyze datasets with millions of records and up to 1,000's of data fields. Cubist has been used for many data-mining and GIS research projects including a process to update the National Land Cover database (Fry, Coan, Homer, Meyer, & Wickham, 2009), to determine local effects of Mercury emissions (Walsh & Milligan, 2005), to predict profit in dairy farms (Yli-heikkil, Tauriainen, & Sulkava, 2015), to estimate vegetation canopy density (Huang, Yang, Wylie, Homer, & Itss, 2001), and to forecast wind farm power production (Wiener, Pearson, Lambi, Myers, & Goodrich, 2011), among many others.

There are some additional characteristics of Cubist that were factors in choosing it for this research. There is an open source version of the code available. This allows us to analyze and modify the code freely should the need arise. While open source code is available, Cubist is a proprietary product produced by a company called RuleQuest. This company sells and supports a version of the algorithm designed to work on

multiple processors. Paid professional support is often a requirement for larger organizations when adopting new tools and software.

Another advantage to Cubist is the structure of the data models that it creates. These models are well-structured plain text rules. This means that the models can easily be opened up in a text editor and the rules analyzed by a human, they could then be applied by hand if so desired.

```
rules="98"
conds="3" cover="68950" mean="0.1" loval="0" hival="100" esterr="0.2"
type="2" att="l3b1" cut="42" result="<="
type="2" att="l3b2" cut="35" result=">"
type="2" att="l1b3" cut="45" result=">"
coeff="1" att="l1b2" coeff="0.01" att="l1b3" coeff="-0.013" att="l1b6" coeff="-
0.008" att="y|1b2" coeff="0.02" att="l2b3" coeff="-0.03" att="l2b4" coeff="0.02"
att="l4b3" coeff="0.01" att="l4b4" coeff="-0.017" att="l4b5" coeff="-0.01"
att="l4b6" coeff="0.01" att="l5b5" coeff="-0.02" att="l5b6" coeff="0.02"
att="l6b2" coeff="-0.02" att="l6b5" coeff="0.013"
```

Figure 2 – Sample Cubist rule. This rule has three condition that must be met. The first band of the third scene in the Landsat time series stack must be less than or equal to 42 while the second band of the third scene must be greater than 35 and finally, the third band of the first scene must be greater than 45. If these conditions are met, then, then the linear model equation using the listed coefficients and band values are used to calculate the result for each pixel

Finally, the Cubist algorithm is fast. It is written in the C programming language and is very fast and efficient when building and applying models. In fact, Cubist is the fastest part of the toolset built for this research. Extracting Landsat data from the hard drive and pulling it into the Cubist algorithm is the biggest bottleneck with this tool.

Methods

This section describes the detailed steps I performed in this research, from data collection and processing, to result analysis and verification. Most of the steps were

scripted using Python and rely heavily on the Geospatial Data Abstraction Library (GDAL). Open source tools were chosen whenever possible to facilitate low-cost repeatability of this research. Most of the scripts I developed are available in the appendices. I processed the data on a distributed number of computers. The code is separated in such a way that the analysis can be scaled in parallel across numerous machines.

Study Area

I used Landsat 5 scenes as the remotely sensed data for this project. I collected Landsat data individually by scene, with two scenes per year, for the years 2009 through 2011. Each scene covers an area defined by the Worldwide Reference System (WRS) path/row boundaries.

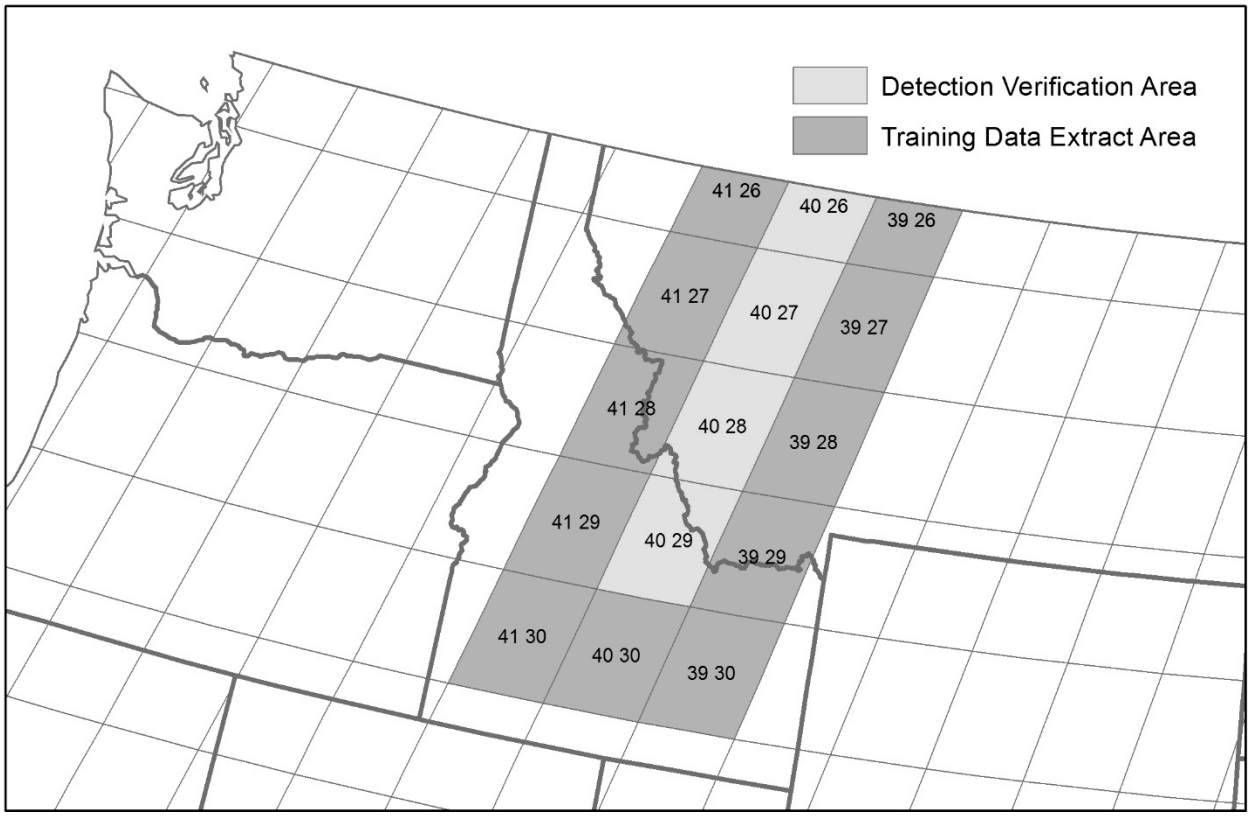

Figure 3 – Study area covering Western Montana and central Idaho. Fire detections were verified in path 40, rows 26 through 29. Data used to train the model for detection scenes was collected from the neighboring scenes.

orientation and 183 kilometers along its east-west orientation. Each scene is approximately 31,000 square kilometers in area. I collected pre-processed Landsat data covering paths 39 through 41 and rows 26 through 30. This area covers parts of Montana and Idaho (Figure 3). I extracted data from all WRS path/rows in the study area but only applied the models built from that data to path 40, rows 26 through 29. I built training datasets for each detection scene by combining the data from all neighboring training scenes. The area where I verified detections covers an area about 124,000 square kilometers. The time period, or detection window, in which I'm trying to identify fires is between the third and fifth scenes in the stack, which is approximately September 2009 through September 2010.

Prep BARC Data:

I used the Burned Area Reflectance Classification (BARC) datasets to identify fires for labeling disturbance pixels in the dataset extracted for training the Cubist algorithm. The Burned Area Emergency Response (BAER) team produces BARC rasters for dozens of fires across the western United States every year. BARC data uses Landsat's near infrared (NIR) and shortwave-infrared (SWIR) bands to calculate a normalized burn ration (NBR) index suitable for identifying fire severity using the following equation:

$$NBR = \frac{(NIR - SWIR)}{(NIR + SWIR)}$$

The BARC data compares the NBR index from a Landsat scene before the fire and the NBR index from a Landsat scene after the fire. This comparison is called the differenced normalized burned ratio (dNBR). Pixels with high dNBR values are considered the highest severity areas of the fire while the lowest values are considered unburned. The dNBR equation is as follows:

$$dNBR = NBR_{pre-fire} - NBR_{post-fire}$$

The BAER team rescales the dNBR burn severity index to a 0-256 scale. This is then segmented into four classes, representing unburned area, low severity, medium severity and high severity areas. Low severity pixels are defined as areas where surface material is not completely consumed and still recognizable. Canopy and understory vegetation is still usually green. Moderate severity pixels are areas where up to 80 percent of surface material is consumed by fire, vegetation is typically scorched and

appears brown. High severity pixels are areas where all of the surface material is removed and vegetation, where it remains, is typically black in appearance (Parsons, Robichaud, Lewis, Napper, & Clark, 2010).

The BARC data for each fire is evaluated by a BAER field team. Typical class breaks for BARC data are 0 to 75 as unburned, 76 to 109 as low severity, 110 to 187 as medium severity, and 188 to 255 as high severity (Parsons et al., 2010). Adjustments to the class breaks for each fire are adjusted, if necessary, based on the field team's observations (Parsons et al., 2010). I downloaded all BARC data in the training area from the BARC repository. Data formats are sometimes inconsistent in the repository. Most fires include the BARC four class raster, but occasionally it is missing from the BARC repository. In these instances, the BARC 256 data are provided, along with the custom break points that are needed to convert the BARC 256 raster into the BARC four raster. In these instances, I reclassified the BARC 256 raster into a BARC four raster. Additionally, most BARC datasets are in the USGS Albers projection. Occasionally, however, these datasets are provided in a local UTM coordinate system. In these cases, I projected the data into Albers.

After ensuring the BARC data was processed consistently, I calculated the outermost rectangular extent of each BARC fire that would encompass all BARC pixels from that fire. This process identified the minimum extent of the fire in Albers coordinates and inserted the coordinates into a computer file shared by all of my BARC data for each year of interest.

The code for prepping BARC data is available in Appendix III.

Collect and Prepare Fire Perimeter Data

We also needed to extract pixels without a fire disturbance when training our detection model. The BARC dataset is only available for a handful of fires in any given year. In order to minimize the risk of extracting a point as an unburned training pixel that was actually burned, we needed to know where all fires were located. I collected fire perimeters from many national, state and local datasets (Appendix I). These perimeters were combined into a single GIS database. At a bare minimum, I tried to identify the year of the fire. If possible, I also included the start date and containment date of the fire. As much attribute data was kept as possible from the datasets and merged into a consistent schema. I was then able to use this dataset to mask out areas of fire when extracting training data that represented unburned pixels. While this dataset is not a complete historical record, it does reduce significantly the risk of extracting training data incorrectly. This is largely due to the fact that there is much more likely to be an historical fire record and perimeter data for larger fires.

Identify Input Datasets

My training data structure required two Landsat scenes for each year in the analysis. One scene was a minimum vegetation scene and the other was a maximum vegetation scene. The scenes were selected by hand by staff at the USGS for each WRS path/row. Scenes were selected to minimize clouds, snow, and smoke. Surface reflectance values for each Landsat scene were calculated at the USGS using the Landsat

Ecosystem Disturbance Adaptive Processing System (LEDAPS) before the data was downloaded (USGS, 2016). This made the data between scenes more consistent by minimizing variability due to atmospheric conditions. The thermal band from Landsat was stripped from these scenes. In its place is a mask band that masks clouds, water, snow, ice, and smoke. These steps were all performed by personnel or by automated processes at the USGS.

Extract Training Data

For every WRS path/row, I created a dataset to train the Cubist model. This dataset consisted of all the burned pixel data for each of the eight neighboring WRS path/rows as well as a random sampling of unburned pixels. For each WRS path/row in the training dataset, I built a 3-year Landsat time series stack. I used two scenes for each year. One scene was a leaf-off scene and one was a leaf-on scene.

Because the Landsat satellite orbit drifts slightly in space, and because of limited fuel, satellite thrusters are only burned infrequently as the drift becomes relatively severe (NASA, n.d.). Because of this drift, the footprints of scenes from multiple passes of the same WRS path/row do not line up perfectly with each other. For each scene, I created a data mask. I calculated the minimum congruent extent for all six data masks to determine the area from which I could extract training data. This prevented extraction of values with no data from some scenes where data did not overlap.

I performed an intersection between my data mask and each BARC dataset extent for each BARC fire in the year of analysis. When a BARC fire extent intersected

my data mask, I extracted all the pixel data for each Landsat scene where a BARC pixel was classified as either medium or high severity. Additionally, I calculated the Julian day of year for each Landsat scene and included that variable in my dataset. Finally, the last variable in my dataset indicated that the pixel represented a burned location. Burned pixels were given a value of 100. I saved all variables for each Landsat band, Julian day of year, and burned pixel as a single row in a data file. I extracted data into individual data files for each WRS path/row and year combination.

I also extracted data from unburned areas. I randomly selected rows within my scene, and within each row, I randomly selected pixels to extract undisturbed data. For each pixel chosen for extraction, I checked to see if it intersected any of my BARC data or any fire perimeter in my fire perimeters database. If the pixel intersected a fire, and the fire occurred in the year of analysis, and before the scene was taken, I removed the pixel. This minimized training pixels as unburned pixels in areas where fire actually occurred. I limited the number of unburned pixels extracted to the lesser of either 10 times the number of burned pixels extracted, or 1 million pixels. Spectral data variables for unburned pixels were extracted, and Julian day of year was calculated, using the same process for burned pixels. However, the final variable was set to 0 instead of 100 to represented unburned pixels.

The code for the extraction process can be seen in Appendix IV.

Build the Training Model

For each WRS path/row being analyzed, I combined the training data files from all eight neighboring WRS path/rows. I did not include the training data from the WRS path/row being analyzed in order to prevent biasing the results by detecting the same fires used to train the model. I took this combined training data file for each WRS path/row and applied the Cubist statistical algorithm. Cubist built a separate model for each scene being analyzed that could then be used to identify likely fires.

The code for prepping the data and building the models can be seen in Appendix V

Using the Detection Model

The detection process followed similar steps as the extraction process. I extracted the pixel data for each pixel in my 3-year Landsat time series. I also converted the date the scene was recorded by Landsat into the Julian day of year and included that as a variable in my training dataset. However, instead of setting the burned/unburned variable as either 0 or 100, I left it as unknown.

I then applied the Cubist model to every pixel which returned a value between approximately 0 and 110. The higher the value, the more likely the pixel was burned during the detection window. I then took the returned values calculated by the Cubist model and applied them to a new single band likelihood of fire raster.

The code for applying the model to a new scene can be seen in Appendix VI.

Prep the Likelihood of Fire Raster for Analysis

The raster created from my Cubist model represented a likelihood value that each pixel experienced a fire during the year of analysis. I used the Schowengerdt method to set a standard threshold for determining the cutoff between burned and unburned values by adjusting the threshold while looking at overlaid BARC data (Singh, 1989). I set the threshold at the point around where the BARC fires were just distinguishable. This value was typically around 95. I reclassed this raster, setting every pixel with a value equal to, or above 95 as a burned pixel, and everything below as unburned. Finally, I performed a raster sieve operation to remove single pixels from my analysis. This removed a great deal of noise from the output. In this analysis, I only looked for fire that was 2 or more adjoining or diagonal $30m^2$ pixels, or at least 0.5 acres in size. This created my final output raster of pixels where the model had detected fire in the detection window.

The code for preparing the results raster for analysis can be seen in Appendix VII.

Verify the Results

Verification of the results was difficult. We have a good idea when and where many, but not all, fires have occurred by using our history fire point database from the USGS. However, this is only point data and does not provide perimeter delineation, or more importantly, internal heterogeneity of the fire. We can supplement, to an extent, with our custom fire perimeter database, but again, these are just perimeters and don't

reflect the actual severity of the fire over each pixel. Neither of these databases are a complete record of all fires that have occurred over the landscape in our study area and over our study time period. BARC data are the only national, field verified, datasets we have that show internal heterogeneity of the fire, and, because it is our training source, output severity classifications should match the BARC input data if the model is trained perfectly. However, only a handful of fires are processed as BARC datasets by the BAER team. For these reasons, it is difficult to build an automated testing model to verify each detected fire. I opted instead to do a visual analysis of each detection and verify each detection by comparing to high resolution aerial imagery and, when possible, cross-reference with my various historical fire databases. To facilitate the verification, I created a custom web map application (Figure 4)

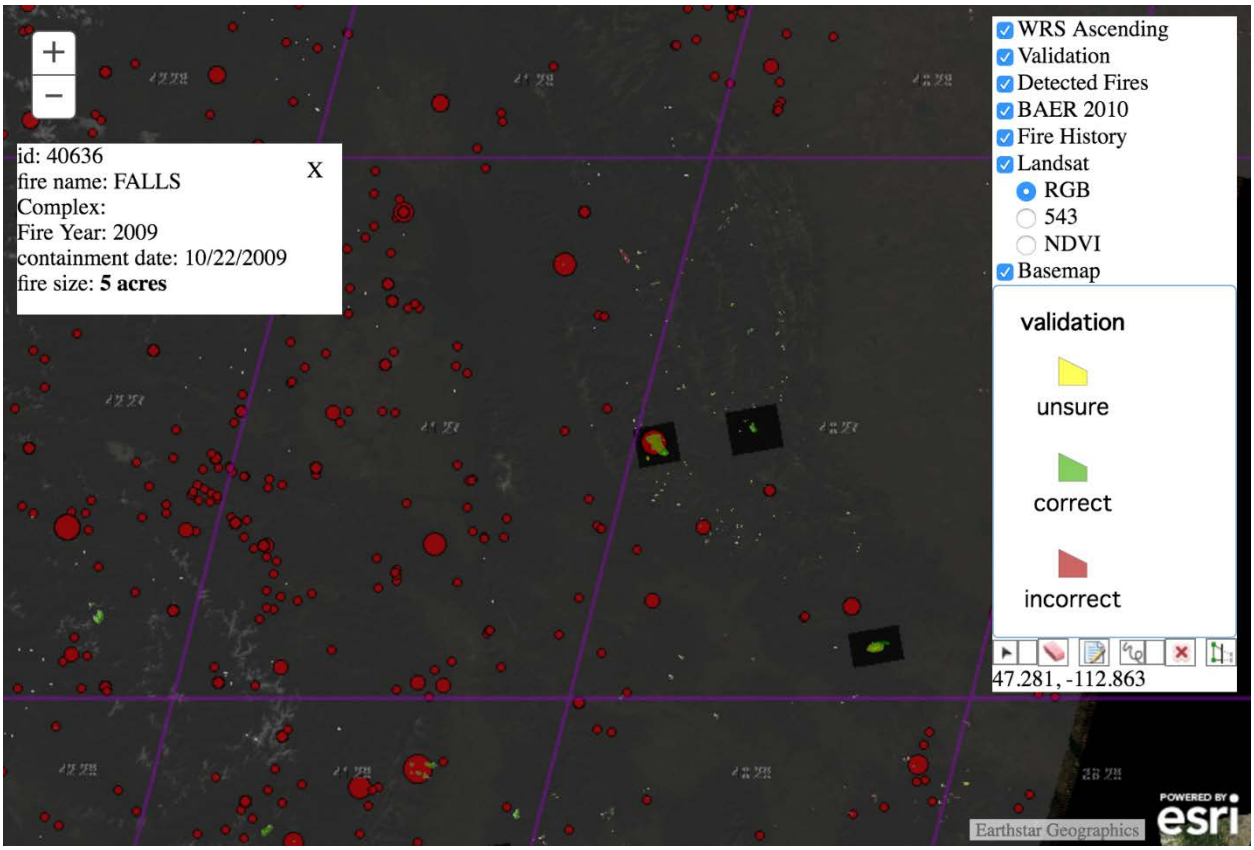

Figure 4 – Validation tool. This tool runs in a standard web browser and can be accessed from any internet connected computer. The tool allows the user to view all detections as well as the Landsat data where the disturbances were detected. Ancillary data, including fire history points with attribute data, BAER team data and Landsat orbital paths can all be toggled on and off allowing the user to quickly determine the validity of a detection. Red dots represent fires in the FOD and are sized based on the total burned area field in the FOD. The rasters in the black boxes are BAER data. The black box is the data mask in the BAER raster. It is set slightly opaque and can be toggled on and off to see the disturbance in the imagery below. The fire information in the top left corner is the attribute information for one of the FOD fires. The user can click on any FOD and see the basic attribute information associated with that fire. The "unsure", "correct", and "incorrect" toggles on the right allow the user to draw polygons on the map for each of those verification classes.

This application included all of the fire detections as well as all of the BARC data and fire history points relevant to my detection time period. Landsat's orbital path and rows could be toggled to orient me to the relevant testing area. The final Landsat scene in my detection time period was available for viewing using several band combinations, including standard RGB, 543 and an NDVI index. This was used to look at the actual data where a disturbance was detected. A high resolution basemap from ESRI was included as well. This basemap was not time-stamped, and the date and time the

imagery was taken varied across the study area. However, this provided another clue when identifying the disturbance.

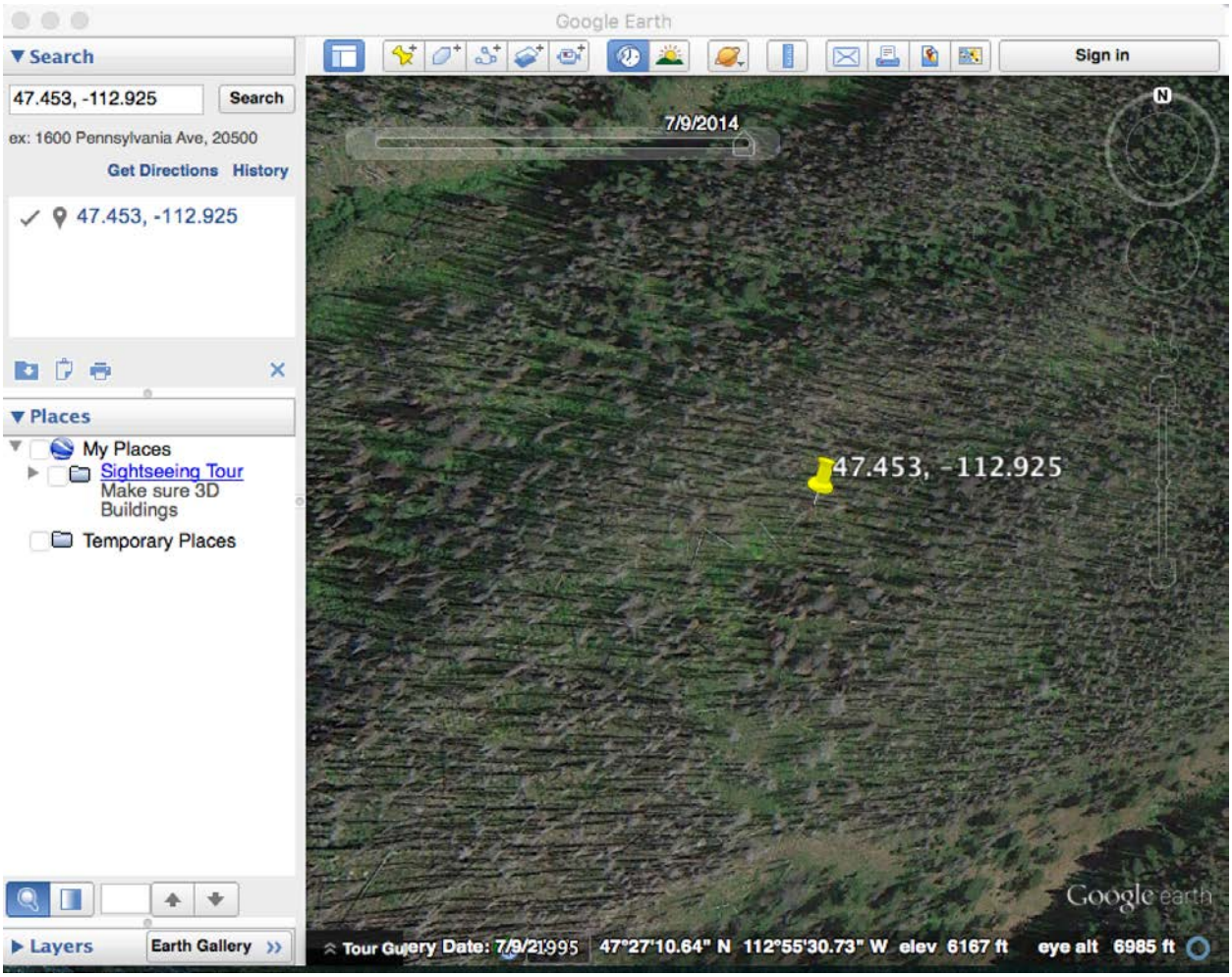

Figure 5 – Google Earth. This screenshot of Google Earth shows one of the detected disturbance locations in high resolution. This was a correctly detected fire. The time slider at the top left corner of the image could be adjusted allowing the user to view all imagery available for different time periods. Looking at imagery before the detected disturbance and after the detected disturbance allowed me to verify the disturbance time period.

When clicking on a historical fire point, attribute data for that fire, including the name of the fire, the fire year, the containment date of the fire (if it existed in the attribute data) and the fire size in acres were displayed in a popup along with a unique integer identifier for that fire. This was used to cross-reference historical fires with my

detected fires.

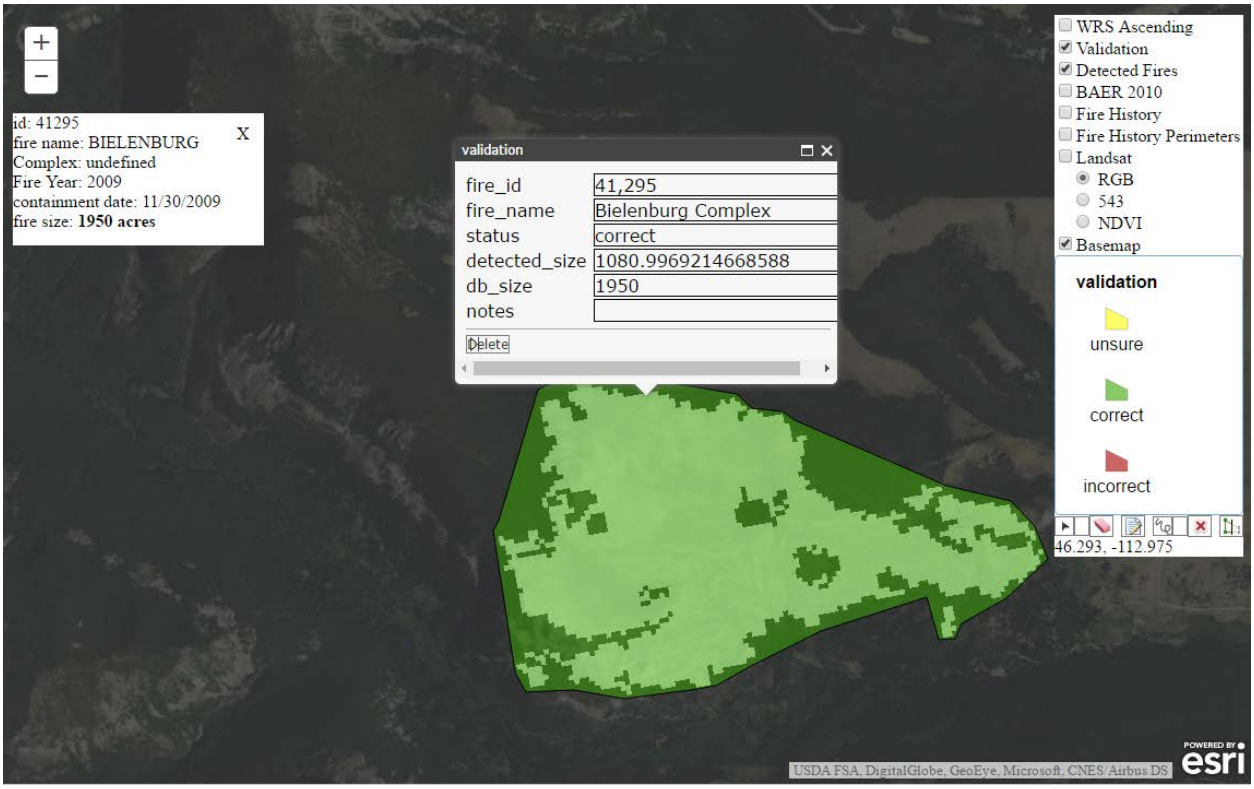

Figure 6 - An example of a correctly detected disturbance in the validation tool. This was the Bielenaburg Complex fire in 2009. The validation box in the center of the screen allowed me to manually enter in known attributes about the detection that I could cross reference with my fire history attributes. Selected fire history attributes could be compared in the top left box. The detected size is automatically calculated from the fire perimeter I drew around each detection.

When clicking on a pixel on the map, the latitude and longitude for that location was displayed. I copied the latitude and longitude values into Google Earth. I then used Google Earth's imagery to try and verify the fire disturbance and time period. I adjusted the imagery time-slider in Google Earth to determine if a fire disturbance occurred within or close to (depending on the dates of the imagery) the detected fire disturbance.

After I verified the disturbance using Google Earth, I drew a polygon around the disturbance in the verification app. I then entered in attribute information about that detection, including whether or not the disturbance could be verified as correct or

incorrect. If I was unable to determine whether or not a verification was correctly detected, I labeled the status as indetermined. There were many times when the spatial or temporal resolution of the aerial imagery was not adequate enough to make a determination. I included a note to provide a description of the details about an incorrect classification or a classification that couldn't be determined.

If I was able to cross-reference the fire with a historical known fire, I entered the unique fire, fire size, and name into the attribute data of the detected fire.

Results

I evaluated all detections in four scenes in my study area. Path 40, rows 26 through 29. There were a total of 881 detections in those four scenes, or approximately 220 detections per scene on average. Each scene was approximately 31,110 square kilometers in area. Of those 881 detections, I was able to verify 331 of them as fire detections during the correct detection time window. 256 were clearly not fires, or fires but outside of the detection time window. Another 224 detections could not be determined clearly as fire or non-fire detections.

Of the incorrect detections, 103, or 40% were agricultural land. 39, or 15% were clear-cut or thinned forests. 15, or 6% were fires, but from an early time period, outside of our detection window. 16, or 6.5% were detections on barren land, often at high altitude areas with perennial snow caps. The last 83, or 32% were not classified.

There were 224 fires that I was unable to verify as correctly or incorrectly detected fire disturbances. Of those 224 detections, 31, or 14% were either clear-cut or thinned forests or fires. 109, or 49% were determined as fire disturbances, but may or may not have occurred before our detection time period window. Inability to identify whether these fires fell within, or close to the correct time period was usually due to inadequate high-resolution imagery around the detection window. 23, or 10% were non-forested lands, usually either grass or shrublands. The spatial and temporal resolution of the high-resolution verification imagery was usually too low to adequately determine whether or not these were correctly detected fire disturbances. Finally, 61, or 27% were not classified. These disturbances usually occurred in areas where the spatial or temporal resolution of the validation imagery was of poor quality, there were obstructions such as shadows in the imagery that made an adequate determination difficult, or it was in a non-forested area where determinations were difficult because of the quick recovery of the vegetation type in low level fires.

Table 2 – Summary table of results.

Determination	Note	Count	Percentage of determination	Percentage of total
Correct		***331***		***40.8%***
Incorrect		***256***		***31.6%***
	Agriculture	*103*	*40.2%*	*11.7%*
	Barren	*16*	*6.3%*	*2%*
	Clear-cut or Thinning	*39*	*15.2%*	*4.8%*
	Wrong time period	*15*	*5.9%*	*1.8%*
	Other	*83*	*32.4%*	*10.2%*
Indetermined		***224***		***27.6%***
	Clear-cut or Thinning or fire	*31*	*13.8%*	*3.8%*

Fire, but possibly wrong time window	*109*	*48.7%*	*13.4%*
Non-forest	*23*	*10.3%*	*2.8%*
Other	*61*	*27.2%*	*7.5%*

The Fire Occurrence Database has 204 fire records for the four scenes in my study area and detection window. Of those 204 records, I was able to match 10 with my fire detections. Those 10 matched detections accounted for 85% of the total acreage burned in that time period according to the FOD database. The other 194 were very small to small fires ranging from a few square meters in size to 300 acres, with a majority of the fires being sub-acre and below the detection threshold.

Table 3 – FOD Comparisons. This table compares the fires I detected to the fires in the FOD database. I was able to match 10 of the fires in the FOD database to the detected fires. This accounted for the majority of fire by area in the study area.

FOD fire detections	***Total Acreage***	***Historical Fire Count***
Detected	*15,919*	*10*
Not Detected	*2,707*	*194*
Total	*18,626*	*204*

I stratified the FOD database by fire size and compared my detections (Table 4). Seven fires were greater than 1,000 acres. Of those seven fires, I detected six. The fire not detected was the Gabe Creek fire. This was a lightning fire discovered July 30th 2009. It's containment date was November 20, 2009 (Karen C. Short, 2015). This fire straddled my detection window The majority of this fire occurred prior to the detection window starting on September 16, 2009 (USDA Forest Service, 2009). I correctly detected 3 of the 7 fires in the 50 to 300-acre size class. Only 1 fire smaller than 50 acres was correctly detected. This was a lightning fire in the Salmon-Challis National Forest. A full list of fires in the FOD greater than one acre in size is available in Appendix II.

Table 4 – FOD detections stratified by size. Most large fires were detected by the detection tool. However, most of the fires in the FOD were very small and below the detection threshold of one half acre. There were no fires in the FOD between 300 and 1,000 acres in size.

Reported Fire Size	Records Detected	# of Records	% Detected
1000+ Acres	6	7	85.8%
50 - 300 Acres	3	7	42.9%
10 – 50 Acres	1	11	9%
5 – 10 Acres	0	5	0%
1 – 5 Acres	0	20	0%
<1 Acre	0	154	0%

Of the 331 fire detections that I was able to verify, only 10 could be associated with fires in the FOD. The other 321 fires were not listed in the FOD. A majority of fires in the FOD are below my detection threshold of half an acre.

Comparing the results from this research with the results from LandTrendr, VCT, and MIICA are difficult for several reason. LandTrendr and VCT are only designed to work in forested environments. The detailed analysis of those algorithms was limited to those land classification types. I ran my tool across all land classification types within my four WRS path/row study area. In order to better compare my results to LandTrendr and VCT, I extracted the vegetation physiognomy from the LANDFIRE existing vegetation layer at the centroid of each detection (Table 5).

Table 5 – Verification by Physiognomy

Physiognomy	*Correct*	*Incorrect*	*Indetermined*	*% Correct*
Agricultural	*0*	*57*	*3*	*0%*
Barren	*4*	*9*	*2*	*31%*
Conifer	*178*	*65*	*130*	*73%*
Developed	*3*	*7*	*8*	*30%*
Grassland	*131*	*36*	*54*	*79%*
Hardwood	*1*	*1*	*1*	*50%*
Open Water	*0*	*1*	*0*	*0%*
Riparian	*1*	*1*	*2*	*50%*
Shrubland	*10*	*17*	*16*	*37%*
Snow-Ice	*0*	*2*	*0*	*0%*
Sparsely Vegetated	*2*	*14*	*6*	*13%*

The two biggest physiognomy classes with detections were conifer and grassland with 73% and 79% correct respectively. Those two classes also had the highest number of verifications that were not conclusively labeled as correct or incorrect. 41% of the indetermined detections in the conifer class were fires that could not be verified in the detection window, while 81% of the indetermined detections in the grassland class were fire where the detection window could not be verified. If higher temporal resolution imagery were available to confirm that all of those fires were actually detected within the detection window, then the percentage of correct detections in the conifer and grassland classes would be 78% and 83% respectively.

LandTrendr is designed to detect multiple disturbance types, as well as stable and recovering areas. However, it is only designed to work in forested areas. LandTrendr had an overall omission rate between approximately 31% and 33% while the commission rate for disturbances was between 41 and 48%. In the LandTrendr

study, 31 total fire disturbances were detected with an omission rate between 10% and 26% (Cohen et al., 2010). No commission rate for fires was provided in the study. LandTrendr currently has a better detection rate of wildland fires in forested areas than my tool, however, the number of wildland fire detections was much lower than in my detection analysis. Furthermore, LandTrendr only detected 5 fires smaller than 1,000 acres, while my tool demonstrated numerous correct detections of small fires (Cohen et al., 2010).

VCT is also designed to detect multiple disturbance types. In VCT's accuracy assessment, however, they did not give accuracy percentages by type. The overall commission error for VCT detections is approximately 70% while the omission error is approximately 60% (Jin et al., 2013). The overall accuracy of VCT in forested areas currently performs better than my algorithm. However, VCT does not perform well in non-forested types, while I have demonstrated moderate success in non-forested areas.

The only MIICA analysis available was a case study on two different WRS path/rows. The results of the MIICA analysis are for all change classes, and not explicitly broken out by disturbance type, so again, we cannot directly compare the results to my research. In both case studies, there was a 0% commission error. However, in the two case studies, the omission errors were 18% and 88% (Huang et al., 2010). While MIICA is very good at reducing errors of commissions, it's detection rate seems spurious.

Discussion

It is difficult to directly compare my results against those of the LandTrendr, VCT, and MIICA algorithms. LandTrendr and VCT only work in forested areas while MIICA's validation analysis was fairly minimal. Additional analysis comparing my tool and these three algorithms would be required to ensure comparisons were done using identical procedures. Regardless, my tool does demonstrate moderate success at detecting wildland fire in both forested and non-forested areas.

I was able to detect a majority of the fire, by area, in the FOD database. However, many of the medium size fires that I would expect to detect were not detected. Further refinements to the training data will likely improve the detection rate. I also detected many fires that I could not identify in the FOD. This suggests that many fires do not have public records and that the FOD is incomplete.

The current training data needs improvement through further iterations of the model to reduce commission errors. The most common commission errors occurred on clear-cut and agriculture land. Clear-cuts can look relatively similar to severe, stand removing fires. I had difficulty at times, determining whether an area with a fire detection was a clear-cut or a fire, even when looking at imagery with a higher spatial resolution then Landsat.

Agricultural land goes through significant spectral changes throughout a given year. This confused the algorithm in several scenes. The error rate can be reduced by

adding the agricultural land and clear-cut forest commission error points back into the training data specifically as "undisturbed" pixels. Increasing the number of fires used for training the model would also help significantly improve the detection rate.

BARC data in some of my training areas were very sparse. Sometimes, I only had one or two BARC fires, if any, for a given training scene. Also, since I was only training off of the moderate and high severity pixels, those pixels were even more limited. The lack of data in some parts of my study area decreased the accuracy of the algorithm in those places. In future iterations, Monitoring in Trends and Burn Severity (MTBS) data can be used to supplement BARC data. MTBS data is calculated for most fires over 1,000 acres in size and classifies pixel severity similar to BARC data. It tends not, however, to be field validated (Eidenshink et al., 2007).

Julian day of year was one of the few non-spectral variables used in this analysis. Pixels in a given area can look quite different depending on the time of year. There is usually less vegetation during winter months compared to summer months. Shadows cast from trees and other objects will vary in direction and intensity based on the time of year. Fire behavior and severity in any given area can look quite different early and late in the season compared to the peak season. Julian day of year for each scene, including the scene in which the fire is first detected can be an important clue when building rule models to detect fire. The minimal amount of BARC data over parts of the study area significantly limits the value of the day of year code for the analysis, as there are relatively few fires to compare in any given region and at any given time. An

increase in the number of training fires would increase the value of the Julian day of year variable for detecting fires.

Compounding events also provided obstacles in determining whether or not a fire detection was correctly detected because of a fire. There are many fire detections that appear to be fire that are clear-cut shortly after. It is difficult to ascertain whether the detection was the initial fire, or the post clear-cut. Sometimes the reverse is also true. Areas are clear-cut, and then the slash is burned. Greater temporal resolution in the high-spatial resolution verification imagery would help more accurately determine if these pixels are correctly being detected because of the fire, or incorrectly detected because of the clear-cut.

There needs to be significant experimentation with training area zones. This project used a rudimentary system to define areas where training data would be extracted. It used training data from neighboring scenes. There are many ways in which ecosystems can be segmented, such as LANDFIRE Zones, EPA Regions, and other ecological boundary systems. Training data divvied up by regions of similar characteristics may lead to better detection models for those same regions.

I would like to repeat this research using Landsat 8 data. Landsat 5, on which this model was trained, is no longer in service, and Landsat 7's problems with the Scan Line Compensator make its data less than ideal for analysis. Landsat 8 was only launched in February 2013, and fully calibrated imagery for analysis was not available until April 2013. There is not yet a rich time series of data available with Landsat 8 for

performing a similar analysis. However, by 2017 or 2018, there should be enough Landsat 8 data available to repeat this experiment. I could also experiment with combining Landsat 5 and Landsat 8 data in a time series. The spectral properties recorded by Landsat 8 for each band differ slightly from Landsat 5 (Table 1), however, the differences may be negligible when applying the fire detection rule models.

Conclusions

This tool shows promise for detecting fires and other disturbances. To be effective, the model needs to go through several more iterations of training both for disturbed pixels and undisturbed pixels. Many of the areas where the model classified a disturbance incorrectly, such as agricultural pixels, should be relatively easy to train out of the model. More fire training data would be extremely beneficial to improving the accuracy of the model and MTBS data may be a good solution. Continued development will be required to turn this into an effective tool ready for production.

Limitations and Future Research

This project represents over 1000 hours of development and testing. I have begun to lay the foundation for a sophisticated machine-learning fire and change detection tool, however, significant more time investment will be required to turn this research into a full-production system. However, the road map for future work on this research is fairly clear. The first step will be to build a better management system for the training data. Once the training data are more optimally organized, the web based

validation tool can be expanded to convert user verifications back into training data, either in the affirmative or the negative. This will allow analysts to quickly inspect and verify disturbance detections. Once the verification is complete, the new data should be integrated back into the training data model and the process rerun. Over time, this will allow human analysts to constantly improve the training data set, enabling the computer to make better detection rules. Over-time, through an iterative machine-learning process we can create an accurate fire detection engine for all of the United States, and elsewhere in the world.

References

Balchin, W. G. V. (1987). United Kingdom Geographers in the Second World War: A Report. *The Geographical Journal*, *153*(2), 159–180. http://doi.org/10.2307/634869

Betker, M. R., Fernando, J. S., & Whalen, S. P. (1997). The History of the Microprocessor. *Bell Labs Technical Journal*, (1947), 29–56. http://doi.org/10.1002/bltj.2082

Cohen, W. B., & Goward, S. N. (2004). Landsat's Role in Ecological Applications of Remote Sensing. *BioScience* , *54* (6), 535–545. http://doi.org/10.1641/0006-3568(2004)054[0535:LRIEAO]2.0.CO;2

Cohen, W. B., Yang, Z., & Kennedy, R. (2010). Detecting trends in forest disturbance and recovery using yearly Landsat time series: 2. TimeSync - Tools for calibration and validation. *Remote Sensing of Environment*, *114*(12), 2911–2924. http://doi.org/10.1016/j.rse.2010.07.010

Dale, V. H., Brown, S., Haeuber, R. A., Hobbs, N. T., Huntly, N., Naiman, R. J., … Valone, T. J. (2000). Ecological Principles and Guidelines for Managing the Use of Land. *Ecological Applications*, *10*(3), 639–670. http://doi.org/10.1890/1051-0761(2000)010[0639:EPAGFM]2.0.CO;2

Downing, T. (2011). *Spies In The Sky: The Secret Battle for Aerial Intelligence During World War II*. Little, Brown Book Group.

Eidenshink, J., Schwind, B., Brewer, K., Zhu, Z., Quayle, B., & Howard, S. (2007). A Project for Monitoring Trends in Burn Severity. *Fire Ecology*, *3*(1), 3–21. http://doi.org/10.4996/fireecology.0301003

Fleming, M. D., Berkebile, J. S., & Hoffer, R. M. (1975). Computer-aided analysis of Landsat-1 MSS data: A comparison of three approaches, including a "modified clustering" approach. *LARS Technical Reports*, 96.

Fry, J. A., Coan, M. J., Homer, C. G., Meyer, D. K., & Wickham, J. D. (2009). *Completion of the National Land Cover Database (NLCD) 1992 – 2001 Land Cover Change Retrofit Product*. *U.S. Geological Survey Open-File Report*. Sioux Falls, SD.

Giglio, L., Csiszar, I., & Justice, C. O. (2006). Global distribution and seasonality of active fires as observed with the Terra and Aqua Moderate Resolution Imaging Spectroradiometer (MODIS) sensors. *Imaging*, *111*(July 1996), 1–12. http://doi.org/10.1029/2005JG000142

Hannavy, J. (2013). *Encyclopedia of Nineteenth-Century Photography*. (J. Hannavy, Ed.) (reprint). New York: Routledge.

Hildebrandt, A. (1908). *Airships Past and Present* (Translated). London: R. Oldenbourg.

Huang, C., Goward, S. N., Masek, J. G., Thomas, N., Zhu, Z., & Vogelmann, J. E. (2010). An automated approach for reconstructing recent forest disturbance history using dense Landsat time series stacks. *Remote Sensing of Environment*, *114*(1), 183–198. http://doi.org/10.1016/j.rse.2009.08.017

Huang, C., Yang, L., Wylie, B., Homer, C., & Itss, R. (2001). A Strategy for Estimating Tree Canopy Density Using Landsat 7 ETM+ and High Resolution Images over Large Areas. *Third International Conference on Geospatial Information in Agriculture and Forestry*, CD–ROM, 1 disk.

Jin, S., Yang, L., Danielson, P., Homer, C., Fry, J., & Xian, G. (2013). A comprehensive change detection method for updating the National Land Cover Database to circa 2011. *Remote Sensing of Environment*, *132*, 159–175. http://doi.org/10.1016/j.rse.2013.01.012

Kennedy, R. E., Yang, Z., & Cohen, W. B. (2010). Detecting trends in forest disturbance and recovery using yearly Landsat time series: 1. LandTrendr - Temporal segmentation algorithms. *Remote Sensing of Environment*, *114*(12), 2897–2910. http://doi.org/10.1016/j.rse.2010.07.008

Kurzweil, R. (2016). Micro Processor Cost per Transistor Cycle. Retrieved February 5, 2016, from http://www.singularity.com/charts/page62.html

Lillesand, T., Kiefer, R. W., & Chipman, J. (2015). *Remote Sensing and Image Interpretation* (7th ed.). Wiley Global Education.

Lillestrand, R. L. (1972). Techniques for Change Detection. *IEEE Transactions on Computers*, *C-21*(7), 654–659. http://doi.org/10.1109/T-C.1972.223570

Masek, J. G., Cohen, W. B., Leckie, D., Wulder, M. A., Vargas, R., De Jong, B., … Smith, W. B. (2011). Recent rates of forest harvest and conversion in North America. *Journal of Geophysical Research: Biogeosciences*, *116*(2). http://doi.org/10.1029/2010JG001471

McGrath, D. (2011). Intel details Sandy Bridge at ISSCC. Retrieved February 5, 2016, from http://www.eetimes.com/document.asp?doc_id=1258749

Mirajkar, M. A., & Srinjvasan, T. R. (n.d.). Landsat photo-interpretation for preparation of small scale soil maps through a multistage approach. *Journal of the Indian Society of Photo-Interpretation*, *3*(2), 87–92. http://doi.org/10.1007/BF02994475

NASA. (n.d.). Landsat 7 Science Data Users Handbook. NASA.

Nelson, B. K. J., Long, D. G., Connot, J. A., Jewell, S., Kimball, S. M., & Survey, U. S. G. (2016). *LANDFIRE 2010 — Updates to the National Dataset to Support Improved Fire and Natural Resource Management*.

Nelson, K. J., Connot, J., Peterson, B., & Martin, C. (2013). The LANDFIRE refresh Strategy: Updating the National dataset. *Fire Ecology*, *9*(2), 80–101. http://doi.org/10.4996/fireecology.090280

Olsen, R. C. (2007). *Remote Sensing from Air and Space*. Bellingham, WA: SPIE - The International Society for Optical Engineering.

Parker, T. J., Clancy, K. M., & Mathiasen, R. L. (2006). Interactions among fire, insects and pathogens in coniferous forests of the interior western United States and Canada. Thomas J. Parker. 2006; Agricultural and Forest Entomology - Wiley InterScience. *Agricultural and Forest Entomology*, *8*(3), 167–189. http://doi.org/10.1111/j.1461-9563.2006.00305.x

Parsons, A., Robichaud, P. R., Lewis, S. A., Napper, C., & Clark, J. T. (2010). *Field guide for mapping post-fire soil burn severity*. *USDA Forest Service General Technical Report*. Fort Collins, CO.

Press, G. (2013). A Very Short History Of Big Data. Retrieved January 1, 2016, from http://www.forbes.com/sites/gilpress/2013/05/09/a-very-short-history-of-big-data/#7b638a9f55da

Quinlan, R. (2016). Data Mining with Cubist. Retrieved January 5, 2016, from https://www.rulequest.com/cubist-info.html

Rollins, M. G. (2009). LANDFIRE: a nationally consistent vegetation, wildland fire, and fuel assessment. *International Journal of Wildland Fire*, *18*, 235–249.

Russell, S., & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall.

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, *3*(3), 210–229. http://doi.org/10.1147/rd.33.0210

Short, K. C. (2014). A spatial database of wildfires in the United States, 1992-2011. *Earth System Science Data*, *6*(1), 1–27. http://doi.org/10.5194/essd-6-1-2014

Short, K. C. (2015). Spatial wildfire occurrence data for the United States, 1992-2013 [FPA_FOD_20150323]. 3rd Edition. Fort Collins, CO. http://doi.org/10.2737/RDS-2013-0009.3

Singh, A. (1989). Digital change detection techniques using remotely-sensed data. *International Journal of Remote Sensing*, *10*(6), 989–1003. http://doi.org/10.1080/01431168908903939

Statistic Brain Research Institute. (2016). Statistic Brain Research Institute. Retrieved February 5, 2016, from http://www.statisticbrain.com/average-cost-of-hard-drive-

storage

Theobald, D., & Romme, W. (2007). Expansion of the US wildland–urban interface. *Landscape and Urban Planning*, *83*(4), 340–354. http://doi.org/10.1016/j.landurbplan.2007.06.002

Tully, J. P. (2013). *The wildland-urban interface in the conterminous United States 2000-2010*. WWU Masters Thesis Collection. Paper 295. http://cedar.wwu.edu/wwuet/295.

U.S. Geological Survey. (2015). Acquisition. Retrieved December 5, 2016, from http://landsat.usgs.gov/tools_acq.php

U.S. Geological Survey. (2016). What are the band designations for the Landsat satellites? Retrieved December 5, 2016, from http://landsat.usgs.gov/band_designations_landsat_satellites.php

USDA Forest Service. (2009). Wildland Fire Update. Flathead National Forest.

USGS. (2016). *Landsat 4-7 Climate Data Record (CDR) Surface Reflectance Product Guide*. Sioux Falls, SD.

Vincent, C. H., Hanson, L. A., & Bjelopera, J. P. (2014). *Federal Land Ownership: Overview and Data*. *Congressional Research Service*. Washington DC.

Vogelmann, J. E., Kost, J. R., Tolk, B., Howard, S., Short, K., Chen, X., … Rollins, M. G. (2011). Monitoring landscape change for LANDFIRE using multi-temporal satellite imagery and ancillary data. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *4*(2), 252–264. http://doi.org/10.1109/JSTARS.2010.2044478

Walsh, K., & Milligan, M. (2005). Data Mining to Determine Local Effects of Mercury Emissions. Younger Laboratories.

Walton, J. T. (2008). Subpixel Urban Land Cover Estimation: Comparing Cubist, Random Forests and Support Vector Regression. *Photogrammetric Engineering & Remote Sensing*, *74*(10), 1213–1222.

Wiener, G., Pearson, J. M., Lambi, B., Myers, W., & Goodrich, K. (2011). An evaluation of different data mining methods for forecasting wind farm power. Boulder, CO: National Center for Atmospheric Research.

Wright, H. E. (1974). Landscape Development, Forest Fires, and Wilderness Management. *Science*, *186*(4163), 487–495.

Yli-heikkil, M., Tauriainen, J., & Sulkava, M. (2015). Predicting the profitability of agricultural enterprises in dairy farming. In *ESANN 2015 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine*

Learning. Bruges.

Appendix I

This table is a partial list of sources of historical fire perimeters

Number of Fires	*Source*
405	*Apache-Sitgreaves National Forests*
359	*Bitterroot National Forest*
2	*Bitterroot National Forest*
86	*Carson National Forest*
322	*Cibola National Forest*
417	*Clearwater National Forest*
366	*Coconino National Forest*
459	*Coronado National Forest*
189	*Flathead National Forest*
53643	*Geospatial Multi-Agency Coordination*
440	*Gila National Forest*
3590	*Idaho Panhandle National Forest*
305	*Kaibab National Forest*
848	*Kootenai National Forest*
130	*Lincoln National Forest*
190	*Lolo National Forest*
554	*Nez Perce-Clearwater National Forests*
131	*Prescott National Forest*
248	*Santa Fe National Forest*
174	*Tonto National Forest*
5440	*US Forest Service Region 1*
340	*US Forest Service Southwestern Region*
237	*Wyoming Wind River Bighorn Basin District*

Appendix II

List of all FOD fires in the study area greater than one acre in size

Fire Name	*Fire Detected*	*Fire Size (acres)*	*Land Owner*	*Discovery Date*	*Containment Date*	*Cause*
Fred Burr Creek	*False*	*1.1*	*Private*	*9/26/2009*	*9/27/2009*	*Miscellaneous*
	False	*1.2*	*Private*	*4/10/2010*	*N/A*	*Debris Burning*
Park Lake	*False*	*1.25*	*USFS*	*7/23/2010*	*9/15/2010*	*Lightning*
	False	*1.3*	*State*	*7/18/2010*	*N/A*	*Miscellaneous*
	False	*2*	*Private*	*7/15/2010*	*N/A*	*Miscellaneous*
	False	*2*	*Private*	*4/25/2010*	*N/A*	*Debris Burning*
Cigarette Fire	*False*	*2*	*Tribal*	*9/16/2009*	*9/16/2009*	*Smoking*
Nelson Fire	*False*	*2*	*Tribal*	*4/4/2010*	*4/4/2010*	*Arson*
Hibbs Fire	*False*	*2*	*Private*	*4/8/2010*	*4/8/2010*	*Railroad*
Marco	*False*	*2*	*BLM*	*8/26/2010*	*8/27/2010*	*Equipment Use*
	False	*3*	*State*	*7/31/2010*	*N/A*	*Lightning*
Wallace Ditch	*False*	*4*	*USFS*	*4/2/2010*	*4/2/2010*	*Debris Burning*
Falls	*False*	*5*	*USFS*	*8/5/2009*	*10/22/2009*	*Lightning*
Bonita Ranger Station Road	*False*	*5*	*Private*	*9/29/2009*	*9/29/2009*	*Fireworks*
	False	*6*	*Private*	*7/16/2010*	*N/A*	*Miscellaneous*
	False	*6*	*Private*	*7/16/2010*	*N/A*	*Railroad*
	False	*7*	*State*	*5/2/2010*	*N/A*	*Debris Burning*
	False	*10*	*Private*	*3/25/2010*	*N/A*	*Debris Burning*
White Calf Fire	*False*	*10*	*Tribal*	*7/19/2010*	*7/19/2010*	*Lightning*
Yellowjacket Creek	*False*	*11*	*USFS*	*8/2/2009*	*10/22/2009*	*Lightning*
	False	*12*	*Private*	*4/17/2010*	*N/A*	*Debris Burning*
Dupont Deuce	*False*	*13*	*Private*	*10/23/2009*	*10/24/2009*	*Debris Burning*
Haynes	*False*	*15*	*USFS*	*3/18/2010*	*4/1/2010*	*Miscellaneous*
	False	*19*	*State*	*8/1/2010*	*N/A*	*Lightning*
Star Falls	*False*	*20*	*USFS*	*9/18/2009*	*9/30/2009*	*Campfire*
Jesse	*True*	*20*	*USFS*	*8/26/2010*	*8/31/2010*	*Lightning*
	False	*25*	*Private*	*4/24/2010*	*N/A*	*Debris Burning*
	False	*49*	*Private*	*9/28/2010*	*N/A*	*Debris Burning*
Upper Railroad	*True*	*70*	*USFS*	*10/3/2010*	*10/25/2010*	*Lightning*
	False	*99*	*Private*	*5/15/2010*	*N/A*	*Miscellaneous*
North Lyons Creek	*False*	*104*	*Private*	*4/28/2010*	*N/A*	*Debris Burning*
Copper Creek	*False*	*109*	*State*	*9/22/2009*	*9/30/2009*	*Miscellaneous*

Macdonald Pass	*True*	*170*	*USFS*	*9/25/2009*	*10/6/2009*	*Miscellaneous*
Pig Fire	*False*	*200*	*Private*	*3/21/2010*	*3/21/2010*	*Debris Burning*
Sand Basin	*True*	*305*	*USFS*	*8/23/2009*	*11/19/2009*	*Lightning*
River Breaks	*True*	*1,862*	*USFS*	*9/28/2010*	*10/14/2010*	*Miscellaneous*
Gabe Creek	*False*	*1,932*	*USFS*	*7/30/2009*	*11/20/2009*	*Lightning*
Bielenburg	*True*	*1,950*	*USFS*	*7/12/2009*	*11/30/2009*	*Lightning*
Davis	*True*	*2,015*	*USFS*	*8/26/2010*	*9/10/2010*	*Miscellaneous*
Lily Lake	*True*	*2,120*	*USFS*	*8/13/2009*	*10/9/2009*	*Lightning*
Cardinal Creek	*True*	*2,127*	*USFS*	*7/25/2010*	*11/9/2010*	*Lightning*
Table Mountain	*True*	*5,280*	*USFS*	*9/13/2009*	*11/19/2009*	*Lightning*

Appendix III

```
import os
import sys
import glob
import ConfigParser
import gdal
import osr

# command line parameters
year = sys.argv[1]

# Load configuration information from settings.ini
config = ConfigParser.ConfigParser()
config.read("settings.ini")

baer_dir = config.get("Directories", "BAER_DATA_DIRECTORY")

data_dir = os.path.join(baer_dir, year)
perimeter_file = open(os.path.join(data_dir, 'perimeter.txt'), 'w')

# USGS Albers Projection
PROJ = 'PROJCS["Albers_Conic_Equal_Area",' \
        'GEOGCS["NAD83",DATUM["North_American_Datum_1983",' \
        'SPHEROID["GRS 1980",6378137,298.257222101,' \
        'AUTHORITY["EPSG","7019"]],' \
        'TOWGS84[0,0,0,0,0,0,0],' \
        'AUTHORITY["EPSG","6269"]],' \
        'PRIMEM["Greenwich",0,' \
        'AUTHORITY["EPSG","8901"]],' \
        'UNIT["degree",0.0174532925199433,' \
        'AUTHORITY["EPSG","9108"]],' \
        'AUTHORITY["EPSG","4269"]],' \
        'PROJECTION["Albers_Conic_Equal_Area"],' \
        'PARAMETER["standard_parallel_1",29.5],' \
        'PARAMETER["standard_parallel_2",45.5],' \
        'PARAMETER["latitude_of_center",23],' \
        'PARAMETER["longitude_of_center",-96],' \
        'PARAMETER["false_easting",0],' \
        'PARAMETER["false_northing",0],UNIT["meters",1]]'

output_srs = osr.SpatialReference(PROJ)

# Find all files that are barc4
for fire in glob.glob(os.path.join(data_dir, '*', '*barc4_alb.img')):
    print fire

    # split file by the "_" separator and iterate through to
    # identify date in file path
    file_path = fire.split('_')
    for date in file_path:
        if date.startswith(str(year)):
            barc4_ds = gdal.Open(fire, 0)

            # Calculate Extent
```

```
            geoTransform = barc4_ds.GetGeoTransform()
            x1 = geoTransform[0]
            x2 = geoTransform[0] + (barc4_ds.RasterXSize * geoTransform[1])
            y1 = geoTransform[3]
            y2 = geoTransform[3] + (barc4_ds.RasterYSize * geoTransform[5])

            # make sure all extents are calculated in the same Projection
            input_srs = osr.SpatialReference()
            input_srs.ImportFromWkt(barc4_ds.GetProjectionRef())
            transform = osr.CoordinateTransformation(input_srs, output_srs)
            corner_nw = transform.TransformPoint(x1, y1)
            corner_sw = transform.TransformPoint(x2, y2)
            corner_nw = (x1, y1)
            corner_sw = (x2, y2)

            # Output information to csv file as file path, extent, date
            perimeter_file.write(
                    fire + "," + str(corner_nw[0]) + "," +
                    str(corner_nw[1]) + "," +
                    str(corner_sw[0]) + "," +
                    str(corner_sw[1]) + "," +
                    date + "\n")

            # We don't want to output the same file multiple times if
            # the date appears in the file path multiple times
            break
perimeter_file.close()
```

Appendix IV

```
import os
import sys
import glob
import math
import random
import ConfigParser
import subprocess
from datetime import date
import random
import gdal
import ogr
import osr
import numpy as np

# USGS Albers Projection
PROJ = 'PROJCS["Albers_Conic_Equal_Area",' \
        'GEOGCS["NAD83",DATUM["North_American_Datum_1983",' \
        'SPHEROID["GRS 1980",6378137,298.257222101,' \
        'AUTHORITY["EPSG","7019"]],' \
        'TOWGS84[0,0,0,0,0,0,0],' \
        'AUTHORITY["EPSG","6269"]],' \
        'PRIMEM["Greenwich",0,' \
        'AUTHORITY["EPSG","8901"]],' \
        'UNIT["degree",0.0174532925199433,' \
        'AUTHORITY["EPSG","9108"]],' \
        'AUTHORITY["EPSG","4269"]],' \
        'PROJECTION["Albers_Conic_Equal_Area"],' \
        'PARAMETER["standard_parallel_1",29.5],' \
        'PARAMETER["standard_parallel_2",45.5],' \
        'PARAMETER["latitude_of_center",23],' \
        'PARAMETER["longitude_of_center",-96],' \
        'PARAMETER["false_easting",0],' \
        'PARAMETER["false_northing",0],UNIT["meters",1]]'
# Command line arguments to determine path, row, and year of data to extract
path = int(sys.argv[1])
row = int(sys.argv[2])
year = int(sys.argv[3])

# Parse configuration information from the settings.ini file
config = ConfigParser.ConfigParser()
config.read("settings.ini")

# Load configuration information from settings.ini
landsat_data_directory = config.get("Directories", "LANDSAT_DATA_DIRECTORY")
baer_dir = config.get("Directories", "BAER_DATA_DIRECTORY")
training_data_output_directory = config.get("Directories",
                                            "TRAINING_DATA_OUTPUT_DIRECTORY")
fire_history_path = config.get("Directories", "FIRE_HISTORY_DIRECTORY")

# Processed Landsat data location
data_dir = os.path.join(landsat_data_directory, "p0" + str(path) +
                        "r0" + str(row), 'HFA')
baer_dir = os.path.join(baer_dir, str(year))
```

```
outfile_path = os.path.join(training_data_output_directory, str(path) + "_" +
                            str(row), str(year))

try:
    os.makedirs(os.path.dirname(outfile_path) + "\\")
except WindowsError:
    # directory already exists or other permissions/disk errors
    pass
except OSError:
    # Other operating system, and windows write errors
    pass

# This is date order, but not necessarily leaf-on/leaf-off order.
# Get three years of data (two scenes per year).  Middle scene represents
# the year from the command line argument and should represent the
# year of disturbance
years = [year-1, year, year+1]

# Only extract data where disturbance raster is one of these values
# Baer data has the following classes.
# We are only likely to pick up class 3 and 4
#   0 = outside perimeter
#   1 = unchanged / very low (Dark Green) | This means the area after the
fire
#       was indistinguishable from pre-fire conditions. This does not always
#       indicate the area did not burn.
#   2 = low severity (Cyan) | This severity class represents areas of surface
#       fire with little change in cover and little mortality of the dominant
#       vegetation.
#   3 = moderate severity (Yellow) | This severity class is between low and
#       high and means there is a mixture of effects on the dominant
#       vegetation.
#   4 = high severity (Red) | This severity class represents areas where the
#       dominant vegetation has high to complete mortality.
disturbance_values = [3, 4]

# Create separate text files for each disturbance type and for undisturbed
data
outfiles = []
for disturbance_value in disturbance_values:
    outfiles.append(open(outfile_path + "_" +
                    str(disturbance_value) + '.txt', 'w'))

# undisturbed values
outfiles.append(open(outfile_path + "_0" + '.txt', 'w'))


# coordinates of all baer pixels
coordinates = set()

# Lists to store rasters and raster metadata
rasters = []
scene_dates = []
days_of_year = []

# Identify Landsat rasters for analysis
for raster_path in glob.glob(os.path.join(data_dir, '*.img.gz')):
```

```
    # Only get the Landsat scene rasters.  We could do something more elegant
    # with regular expressions, but this gets the job done well enough for
now
    if len(os.path.basename(raster_path)) == 27:
        # only get rasters within the year range we are extracting from
        if int(os.path.basename(raster_path)[12:16]) in years:
            # Open rasters while still gzipped.  This is slower to read,
            # but probably faster and cleaner than extracting first
            if not os.path.isfile(raster_path[0:-3]):
                proc = subprocess.Popen(
                    ['7-Zip\\7z.exe', '-o' +
                        os.path.dirname(raster_path), 'e', raster_path],
                    stdout=subprocess.PIPE,
                    shell=True)
                out, err = proc.communicate()
                if err:
                    print raster_path[0:-3]
                    raise SystemExit
                proc, out, err = None, None, None
            rasters.append(gdal.Open(raster_path[0:-3], 0))
            # Extract date of scene from raster file path and store in a list
            scene_dates.append(int(os.path.basename(raster_path)[12:20]))
    raster_path = None

if len(rasters) > 10:
    print ("ERROR: More than 10 Rasters")

    for raster_path in glob.glob(os.path.join(data_dir, '*.img.gz')):
        if os.path.isfile(raster_path[0:-3]):
            os.remove(raster_path[0:-3])

    raise SystemExit

# Calculate the day of year each scene was taken using the scene dates
for scene_date in scene_dates:
    scene_year = int(str(scene_date)[0:4])
    scene_month = int(str(scene_date)[4:6])
    scene_day = int(str(scene_date)[6:8])
    scene_doy = date(scene_year, scene_month, scene_day).timetuple().tm_yday
    days_of_year.append(scene_doy)

# For now we'll assume all rasters in a scene have the same
extents/projections
# and extract the geotransform data from the first raster
# At some point if that changes, then we will have to project into a uniform
# projection (if different) and compute the extent as the area where all
scenes
# have data
scene_geotransform = rasters[0].GetGeoTransform()
scene_upper_left_x = scene_geotransform[0]
scene_pixel_width = scene_geotransform[1]
scene_upper_left_y = scene_geotransform[3]
scene_pixel_height = scene_geotransform[5]
scene_xsize = rasters[0].RasterXSize
scene_ysize = rasters[0].RasterYSize

# loads a list of our rasters extent in the order [left,top,right,bottom]
```

```
extent = [
    scene_upper_left_x,
    scene_upper_left_y,
    scene_upper_left_x + (scene_xsize * scene_pixel_width),
    scene_upper_left_y + (scene_ysize * scene_pixel_height)
    ]

# Create polygon of scene extent to intersect with baer/disturbance
# data to identify disturbances overlapping the scenes
scene_geometry = ogr.Geometry(ogr.wkbPolygon)
ring = ogr.Geometry(ogr.wkbLinearRing)
ring.AddPoint(extent[0], extent[3])  # x1y1
ring.AddPoint(extent[2], extent[3])  # x2y2
ring.AddPoint(extent[2], extent[1])  # x2y1
ring.AddPoint(extent[0], extent[1])  # x1y2
ring.CloseRings()
scene_geometry.AddGeometry(ring)
ring = None

shp_driver = ogr.GetDriverByName("ESRI Shapefile")
fire_history_ds = shp_driver.Open(os.path.join(fire_history_path,
                                               str(year) + '.shp'))
print os.path.join(fire_history_path, str(year) + '.shp')
if fire_history_ds is None:
    print 'could not open fire history perimeters %s'
    % os.path.join(fire_history_path, str(year) + '.shp')
    raise SystemExit

fire_geometries = []
fire_history_lyr = fire_history_ds.GetLayer()
fire_history_lyr.SetSpatialFilter(scene_geometry)


# numpy array to mask out disturbed pixels which we'll use to then extract a
# sample of undisturbed pixels
mask = np.ones((scene_xsize, scene_ysize))

# Keeps track of the number of disturbed pixels
count = 0

# Read in fire extents and see if they intersect the scene geometry
with open(os.path.join(baer_dir, 'perimeter.txt')) as f:
    for line in f:
        # last character is a newline character, so we trim that off
        # and split into a comma separated list
        baer = line[:-1].split(',')
        baer_file_path = baer[0]
        baer_left = float(baer[1])
        baer_top = float(baer[2])
        baer_right = float(baer[3])
        baer_bottom = float(baer[4])
        baer_date = int(baer[5])  # yyyymmdd
        baer = None
```

```
# Check if fire occurs between 3rd and 5th Landsat scenes in the
# stack
#  ____   ____   ____   ____   ____   ____
# | y1 | | y1 | | y2 | | y2 | | y3 | | y3 |
# | l1 | | l2 | | l1 | | l2 | | L1 | | l2 |
# |____| |____| |____| |____| |____| |____|
#                     ^      ^
#                     | fire |
#########################################################################

if baer_date > scene_dates[2] and baer_date < scene_dates[4]:
    data = []

    # Create polygon of the baer/disturbance extent
    baer_geometry = ogr.Geometry(ogr.wkbPolygon)
    ring = ogr.Geometry(ogr.wkbLinearRing)
    ring.AddPoint(baer_left, baer_bottom)  # x1y1
    ring.AddPoint(baer_right, baer_bottom)  # x2y1
    ring.AddPoint(baer_right, baer_top)  # x2y2
    ring.AddPoint(baer_left, baer_top)  # x1y2
    ring.CloseRings()
    baer_geometry.AddGeometry(ring)
    ring = None

    # If the baer/disturbance geometry intersects the scene geometry
    # then the scene likely has been disturbed so extract disturbed
    # pixels
    if scene_geometry.Intersects(baer_geometry):
        # Open baer/disturbance raster as read only
        baer_ds = gdal.Open(baer_file_path, 0)
        baer_geotransform = baer_ds.GetGeoTransform()
        baer_upper_left_x = baer_geotransform[0]
        baer_pixel_width = baer_geotransform[1]
        baer_we_rotation = baer_geotransform[2]
        baer_upper_left_y = baer_geotransform[3]
        baer_ns_rotation = baer_geotransform[4]
        baer_pixel_height = baer_geotransform[5]
        baer_xsize = baer_ds.RasterXSize
        baer_ysize = baer_ds.RasterYSize

        # Sometimes BAER data is in UTM projection instead of Albers
        # projection
        # Project BAER data into Albers, if necessary, using Nearest
        # projection Neighbor algorithm
        if "utm" in baer_file_path:
            print 'utm'
            # Define transformation from input projection to output
            # projection
            utm_proj = osr.SpatialReference(baer_ds.GetProjection())
            alb_proj = osr.SpatialReference(PROJ)
            transformation = osr.CoordinateTransformation(utm_proj,
                                                          alb_proj)

            # Calculate geotransform of projected raster
            # top left corner
            (ulx, uly, ulz) = transformation.TransformPoint(
                baer_upper_left_x, baer_upper_left_y)
```

```
# bottom right corner
(lrx, lry, lrz) = transformation.TransformPoint(
    baer_upper_left_x + baer_pixel_width*baer_xsize,
    baer_upper_left_y + baer_pixel_height*baer_ysize)
# bottom left corner
(llx, lly, llz) = transformation.TransformPoint(
    baer_upper_left_x, baer_upper_left_y +
    baer_pixel_height * baer_ysize)
# top right corner
(urx, ury, urz) = transformation.TransformPoint(
    baer_upper_left_x + baer_pixel_width * baer_xsize,
    baer_upper_left_y)

transformation = None

# Identify the furthest left point and the top most point
# Sometimes the rasters in a particular projection are
# skewed relative to the new position in a way that makes
# the bottom left corner "more left" than the top corner,
# and similar with the top and bottom most values.  By
# detecting the left most and top most corners, we ensure
# that we output a new projected raster with the correct
# dimensions.
lx = min(llx, ulx)
rx = max(urx, lrx)
ty = max(uly, lly)
by = min(lry, ury)

# Create empty raster in new projection
# with the correct bounds
mem = gdal.GetDriverByName('MEM')
dest = mem.Create('', int(
                      (rx - lx)/baer_pixel_width), int(
    (by - ty) / baer_pixel_height), 1,
    baer_ds.GetRasterBand(1).DataType)

new_geotransform = (int(lx), baer_pixel_width,
                    baer_we_rotation, int(ty),
                    baer_ns_rotation, baer_pixel_height)
dest.SetGeoTransform(new_geotransform)
new_geotransform = None
lr, rx, ty, by = None, None, None, None
dest.SetProjection(alb_proj.ExportToWkt())

# Reproject the image from the original dataset into the
# new dataset using Nearest Neighbor. We use nearest
# neighbor because it is most important to preserve
# actual pixel values rather than altering them with an
# algorithm that "averages" values
gdal.ReprojectImage(baer_ds, dest,
                    utm_proj.ExportToWkt(),
                    alb_proj.ExportToWkt(),
                    gdal.GRA_NearestNeighbour)
utm_proj, alb_proj = None, None

baer_ds = dest
baer_geotransform = baer_ds.GetGeoTransform()
```

```
        baer_upper_left_x = baer_geotransform[0]
        baer_pixel_width = baer_geotransform[1]
        baer_we_rotation = baer_geotransform[2]
        baer_upper_left_y = baer_geotransform[3]
        baer_ns_rotation = baer_geotransform[4]
        baer_pixel_height = baer_geotransform[5]
        baer_xsize = baer_ds.RasterXSize
        baer_ysize = baer_ds.RasterYSize

        # The geometry of the new reprojected raster extent is
        # different and so needs to be recalculated
        baer_geometry = None
        baer_geometry = ogr.Geometry(ogr.wkbPolygon)
        ring = ogr.Geometry(ogr.wkbLinearRing)
        ring.AddPoint(ulx, uly)  # x1y1
        ring.AddPoint(urx, ury)  # x2y1
        ring.AddPoint(lrx, lry)  # x2y2
        ring.AddPoint(llx, lly)  # x1y2
        ring.CloseRings()
        baer_geometry.AddGeometry(ring)
        ring, ulx = None, None
        uly, urx = None, None
        ury, lrx = None, None
        lry, llx, lly = None, None, None

    disturbance_extent = scene_geometry.Intersection(
        baer_geometry).GetEnvelope()
    disturbance_min_x = disturbance_extent[0]
    disturbance_max_x = disturbance_extent[1]
    disturbance_min_y = disturbance_extent[2]
    disturbance_max_y = disturbance_extent[3]

    # Values to extract from Scene Data
    # NOTE: BAER imagery is not always snapped to LANDSAT
    # imagery. This can cause a 1 pixel shift between baer values
    # and underlying Landsat values. Since BAER data is for fires
    # over 40 acres, a few edge pixels shouldn't make too big of
    # a difference (sensitivity analysis)?  This shouldn't cause
    # a problem detecting smaller fires.
    if (abs((scene_upper_left_x - baer_upper_left_x) %
            scene_pixel_width) < (scene_pixel_width / 2)):
        x_snap_factor = 1
    else:
        x_snap_factor = 0

    if (abs((scene_upper_left_y - baer_upper_left_y) %
            scene_pixel_height) < (scene_pixel_height/2)):
        y_snap_factor = -1
    else:
        y_snap_factor = 0

    print "snap factors: ", x_snap_factor, y_snap_factor

    # Subtract 1 from both x_size and y_size since we are
    # rounding  up the starting locations (x_start, y_start)
    # using the math.ceil function.  This ensures that we don't
    # run out of bounds on the East side of our Landsat rasters
```

```
ls_x_start = int((disturbance_min_x - scene_upper_left_x) /
                 scene_pixel_width) + x_snap_factor
ls_x_size = int((disturbance_max_x - disturbance_min_x) /
                scene_pixel_width) - x_snap_factor
ls_y_start = int((disturbance_max_y - scene_upper_left_y) /
                 scene_pixel_height) + y_snap_factor
ls_y_size = int((disturbance_min_y - disturbance_max_y) /
                scene_pixel_height) - y_snap_factor
for raster in rasters:
    for band in range(raster.RasterCount):
        data.append(raster.GetRasterBand(band+1).ReadAsArray(
            ls_x_start, ls_y_start, ls_x_size, ls_y_size))

# Values to extract from BAER
baer_x_start = int((disturbance_min_x - baer_upper_left_x) /
                   baer_pixel_width)
baer_x_size = int((disturbance_max_x - disturbance_min_x) /
                  baer_pixel_width)
baer_y_start = int((disturbance_max_y - baer_upper_left_y) /
                   baer_pixel_height)
baer_y_size = int((disturbance_min_y - disturbance_max_y) /
                  baer_pixel_height)

if baer_x_start < 0:
    baer_x_start = 0
    baer_x_size -= 1

if baer_y_start < 0:
    baer_y_start = 0
    baer_y_size -= 1

data.append(baer_ds.GetRasterBand(1).ReadAsArray(
    baer_x_start, baer_y_start, baer_x_size, baer_y_size))

for column in xrange(min(ls_x_size, baer_x_size)-1):
    for row in xrange(min(ls_y_size, baer_y_size)-1):
        l = []

        x = str(scene_upper_left_x +
                (0.5 * scene_pixel_width) +
                ((ls_x_start + column) * scene_pixel_width))

        y = str(scene_upper_left_y +
                (0.5 * scene_pixel_height) +
                ((ls_y_start + row) * scene_pixel_height))

        l.append(x)
        l.append(y)

        coordinates.add((int(float(x)), int(float(y))))

        for raster in xrange(len(data) - 1):
            l.append(str(data[raster][row][column]))

        for doy in days_of_year:
            l.append(str(doy))
```

```
                        l.append(str(data[len(data) - 1][row][column]))

                        # If any pixel has no value in the Landsat stack,
                        # then ignore the entire pixel stack
                        if all(i != '0' for i in l[2:-6]) and (
                                int(l[-1:][0]) in disturbance_values):
                            outfiles[disturbance_values.index(
                                int(l[-1:][0]))].write((',').join(l) + '\n')
                            count += 1

                        l = None

                    baer_x_start = None
                    baer_x_size = None
                    baer_y_start = None
                    baer_y_size = None

                data = None

            baer_ds = None
            baer_date = None
            baer_file_path = None
            baer_left = None
            baer_top = None
            baer_right = None
            baer_bottom = None
            baer_date = None

        line = None

    # read undisturbed points
    max_undisturbed = min(count*10, 1000000)
    x_size = rasters[0].RasterXSize
    y_size = rasters[0].RasterYSize

    pixels = x_size * y_size
    try:
        offset = pixels / max_undisturbed
    except ZeroDivisionError:
        print 'no pixels to extract'
        for outfile in outfiles:
            outfile.close()

        rasters = None

        # Don't process undisturbed scenes with no disturbance pixels
        raise SystemExit

    # read one line at a time, take pixel sample by offset
    for row in random.sample(range(0, y_size), int(math.sqrt(max_undisturbed))):

        data = []
        for raster in rasters:
            for band in range(raster.RasterCount):
                data.append(raster.GetRasterBand(band+1).ReadAsArray(
                    0, row, x_size, 1))
```

```
    for column in random.sample(range(0, x_size),
                                int(math.sqrt(max_undisturbed))):

        x = str(scene_geotransform[0] + (0.5 * scene_geotransform[1]) +
                (column * scene_geotransform[1]))
        y = str(scene_geotransform[3] + (0.5 * scene_geotransform[5]) +
                (row * scene_geotransform[5]))
        if (int(float(x)), int(float(y))) in coordinates:
            continue

        in_perimeter = False
        fire_history_lyr.ResetReading()
        for fire_perimeter in fire_history_lyr:
            geom = fire_perimeter.GetGeometryRef()
            pt = ogr.Geometry(ogr.wkbPoint)
            pt.SetPoint(0, float(x), float(y))
            # test
            if pt.Within(geom):
                in_perimeter = True
                break

        if in_perimeter is True:
            continue

        l = []
        l.append(x)
        l.append(y)

        for raster in xrange(len(data)):
            l.append(str(data[raster][0][column]))

        for doy in days_of_year:
            l.append(str(doy))

        l.append('0')  # undisturbed class
        if all(i != '0' for i in l[2:-7]):
            outfiles[-1].write((',').join(l) + '\n')

for outfile in outfiles:
    outfile.close()
rasters = None
```

Appendix V

```
import os
import sys
import glob
import shutil
import csv
import ConfigParser

# Load configuration information from settings.ini
config = ConfigParser.ConfigParser()
config.read("settings.ini")

training_data_path = config.get("Directories",
                                "TRAINING_DATA_OUTPUT_DIRECTORY")

output_path = config.get("Directories",
                         "COMBINED_TRAINING_DATA_OUTPUT_DIRECTORY")

c5_exe = config.get("Executables", "C5_EXE")
cubist_exe = config.get("Executables", "CUBIST_EXE")

c5_names_template = config.get("Templates", "C5_NAMES_TEMPLATE")
cubist_names_template = config.get("Templates", "CUBIST_NAMES_TEMPLATE")

years = [2008, 2009, 2010]
scenes = []

# We can really add a lot of options in a more robust manner here
# but this will work for now.
if "-x8" in sys.argv:
    path = int(sys.argv[2])
    row = int(sys.argv[3])
    output_folder = str(path) + "_" + str(row)

    for year in years:
        scenes.append([path+1, row, year])
    for year in years:
        scenes.append([path+1, row+1, year])
    for year in years:
        scenes.append([path+1, row-1, year])
    for year in years:
        scenes.append([path, row+1, year])
    for year in years:
        scenes.append([path, row-1, year])
    for year in years:
        scenes.append([path-1, row, year])
    for year in years:
        scenes.append([path-1, row+1, year])
    for year in years:
        scenes.append([path-1, row-1, year])

else:
    raise SystemExit
```

```
output_path = os.path.join(output_path, output_folder)
os.mkdir(output_path)
shutil.copyfile(c5_names_template, os.path.join(output_path,
'train_c5.names'))
shutil.copyfile(cubist_names_template, os.path.join(output_path,
                                                    'train_cubist.names'))

c5_outfile = open(os.path.join(output_path, 'train_c5.data'), 'w')

for scene in scenes:
    print scene
    inpath = os.path.join(training_data_path, str(scene[0]) + "_" +
                          str(scene[1]), str(scene[2]))

    for file in glob.glob(inpath + '_*.txt'):
        with open(file) as infile:
            c5_outfile.write(infile.read())
print inpath
c5_outfile.close()

with open(os.path.join(output_path, 'train_c5.data'), "rb") as infile, open(
        os.path.join(output_path, 'train_cubist.data'), "wb") as outfile:
    r = csv.reader(infile)

    for row in r:
        if row[-1] != '0':
            row[-1] = "100"
        outfile.write(",".join(row))
        outfile.write("\n")

os.system(cubist_exe + ' -f ' + os.path.join(output_path, 'train_cubist'))
```

Appendix VI

```
# applyTree
# applytree.py [path] [row] [year]
# apply tree to set of scenes

import os
import sys
import glob
import shutil
import struct
import subprocess
import ConfigParser
import numpy as np
from datetime import date
import gdal


# http://stackoverflow.com/a/4914089
def slices(s, *args):
    position = 0
    for length in args:
        yield s[position:position + length]
        position += length

path = int(sys.argv[1])
row = int(sys.argv[2])
year = int(sys.argv[3])

# Load configuration information from settings.ini
config = ConfigParser.ConfigParser()
config.read("settings.ini")

landsat_data_path = config.get("Directories", "LANDSAT_DATA_DIRECTORY")
combined_training_data_output_directory = config.get(
        "Directories", "COMBINED_TRAINING_DATA_OUTPUT_DIRECTORY")
cubist_interpreter_exe = config.get("Executables", "CUBIST_INTERPRETER_EXE")

# for each row in set of rasters
#   export row data to text file of cases
#   process cases with c5
#   import results into new raster

# Processed Landsat data location
data_dir = os.path.join(landsat_data_path, "p0" + str(path) + "r0" +
                        str(row), 'HFA')

# Output location
output_dir = os.path.join(combined_training_data_output_directory,
                          str(path) + "_" + str(row))
output_ds_path = os.path.join(combined_training_data_output_directory,
                              str(path) + "_" + str(row), str(year) + ".img")

try:
    os.makedirs(output_dir)
```

```
except WindowsError:
    # directory already exists
    pass

# This is date order, but not necessarily leaf-on/leaf-off order.
# Get three years of data (two scenes per year).  Middle scene represents
# the year from the command line argument and should represent the
# year of disturbance.  This needs to match the years extracted as part
# of extract.py
years = [year-1, year, year+1]

scene_dates = []
days_of_year = []

# Identify Landsat rasters for analysis
rasters = []
for raster_path in glob.glob(os.path.join(data_dir, '*.img.gz')):
    # Only get the Landsat scene rasters.  We could do something more elegant
    # with regular expressions, but this gets the job done well enough
    if len(os.path.basename(raster_path)) == 27:
        # only get rasters within the year range we are extracting from
        if int(os.path.basename(raster_path)[12:16]) in years:
            # Open rasters while still gzipped.  This is slower to read, but
            # probably faster and cleaner than extracting first
            if not os.path.isfile(raster_path[0:-3]):
                proc = subprocess.Popen(
                    ['7-Zip\\7z.exe', '-o' +
                              os.path.dirname(raster_path), 'e',
                         raster_path], stdout=subprocess.PIPE, shell=True)
                out, err = proc.communicate()
                if err:
                    print raster_path[0:-3]
                    raise SystemExit
                proc, out, err = None, None, None
            rasters.append(gdal.Open(raster_path[0:-3], 0))
            scene_dates.append(int(os.path.basename(raster_path)[12:20]))

if len(rasters) > 10:
    print ("ERROR: More than 10 Rasters")
    raise SystemExit

for scene_date in scene_dates:
    year = int(str(scene_date)[0:4])
    month = int(str(scene_date)[4:6])
    day = int(str(scene_date)[6:8])
    doy = date(year, month, day).timetuple().tm_yday
    days_of_year.append(doy)
    scene_date, year, month, day, doy = None, None, None, None, None

# Get raster properties
# This assumes all rasters have identical extents and projections
driver = rasters[0].GetDriver()
geoTransform = rasters[0].GetGeoTransform()
projection = rasters[0].GetProjection()
xsize = rasters[0].RasterXSize
ysize = rasters[0].RasterYSize
```

```
# Open output raster
output_ds = driver.Create(output_ds_path, xsize, ysize, 1, gdal.GDT_Byte)
output_ds.SetGeoTransform(geoTransform)
output_ds.SetProjection(projection)
output_band = output_ds.GetRasterBand(1)

# Process raster data
filestem = os.path.join(output_dir, 'train_cubist')

# Create day of year string outside of loop, since it should be constant
doys = []
for doy in days_of_year:
    doys.append(str(doy))
doys_str = (',').join(doys)

# Get number of datapoints
datapoints = 0
for raster in rasters:
    datapoints += raster.RasterCount

# The first two datapoints are the coordinates
datapoints += 2

# The second to last set of datapoints are place holders for the year
datapoints += len(days_of_year)

# The last datapoint is our unknown disturbance value labeled "?"
datapoints += 1

for row in xrange(0, ysize, 100):
    sys.stdout.write("%d\\%d  \r" % (row, ysize))

    if ((row + 100) > ysize):
        num_rows = ysize - row
    else:
        num_rows = 100

    data = np.zeros((num_rows * xsize * datapoints), dtype="S10")
    coordsX = np.zeros((num_rows * xsize), dtype="S10")
    coordsY = np.zeros((num_rows * xsize), dtype="S10")

    # Generate the coordinate values.
    # TODO: Put directly into our output array instead of an intermediate
    # array
    i = 0
    for r in xrange(num_rows):
        for column in xrange(xsize):
            coordsX[i] = str(geoTransform[0] + (0.5 * geoTransform[1]) +
                             (column * geoTransform[1]))
            coordsY[i] = str(geoTransform[3] + (0.5 * geoTransform[5]) +
                             ((row + r) * geoTransform[5]))
            i += 1

    data[0::datapoints] = coordsX
    data[1::datapoints] = coordsY

    # Start at position 2 because pos 0 and 1 are the coords
```

```
    position = 2
    for raster in rasters:
        for band in range(raster.RasterCount):
            data[position::datapoints] = raster.GetRasterBand(
                band+1).ReadAsArray(0, row, xsize, num_rows).flatten()
            position += 1

    for doy in days_of_year:
        data[position::datapoints] = np.array([doy])
        position += 1

    data[position::datapoints] = np.array(["?"])

    np.savetxt(filestem + '.cases', data.reshape(
        (num_rows * xsize, datapoints)), "%s", delimiter=",")

    proc = subprocess.Popen([cubist_interpreter_exe, '-f', filestem],
                            stdout=subprocess.PIPE, shell=True)
    out, err = proc.communicate()

    outline = ''
    count = 0
    lines = out.split('\n')
    for line in lines:
        if line != "predicted values:" and line != "":
            outline += struct.pack('B', int(float(line)))
        count += 1

    output_band.WriteRaster(0, row, xsize, num_rows, outline)

rasters = None

output_band.FlushCache()
output_band = None
output_ds = None
```

Appendix VII

```
import os
import sys
import glob
import ConfigParser
import subprocess
import ogr
import gdal
import numpy as np

# Get command line variables
path = int(sys.argv[1])
row = int(sys.argv[2])
year = int(sys.argv[3])

# Load configuration information from settings.ini
config = ConfigParser.ConfigParser()
config.read("settings.ini")

combined_training_data_output_directory = config.get(
        "Directories", "COMBINED_TRAINING_DATA_OUTPUT_DIRECTORY")
landsat_data_directory = config.get("Directories", "LANDSAT_DATA_DIRECTORY")
data_dir = os.path.join(landsat_data_directory,
                        "p0" + str(path) + "r0" + str(row), 'HFA')

################################################################################
##
# Create data mask where all input Landsat pixels are not NoData
################################################################################
##

def makeDataMask():

    first_ls_ds = gdal.Open('/vsigzip/' + glob.glob(
            os.path.join(data_dir, '*.img.gz'))[0], 0)
    driver = first_ls_ds.GetDriver()
    landsat_xsize = first_ls_ds.RasterXSize
    landsat_ysize = first_ls_ds.RasterYSize
    geoTransform = first_ls_ds.GetGeoTransform()
    proj = first_ls_ds.GetProjection()

    data_mask_data = np.ones([landsat_ysize, landsat_xsize])

    for landsat_path in glob.glob(os.path.join(data_dir, '*.img.gz')):
        print landsat_path
        landsat_ds = gdal.Open('/vsigzip/' + landsat_path, 0)
        # Assume NoData if band 1 is 0.  This is probably always, mostly true
        landsat_data = landsat_ds.GetRasterBand(1).ReadAsArray(0, 0)
        landsat_data[landsat_data > 0] = 1
        data_mask_data = data_mask_data * landsat_data

    data_mask_path = os.path.join(combined_training_data_output_directory,
                                  str(path) + "_" + str(row),
                                  str(year) + "_data_mask.img")
```

```
    data_mask_ds = driver.Create(
            data_mask_path, landsat_xsize, landsat_ysize, 1, 1)
    data_mask_ds.SetGeoTransform(geoTransform)
    data_mask_ds.SetProjection(proj)
    data_mask_ds.GetRasterBand(1).WriteArray(data_mask_data)

    # This is not a pixel perfect vector as a lot of interpolation happens,
    # but it is good enough for our purposes of selecting fire points
    output_vect_path = os.path.join(
            combined_training_data_output_directory, str(path) + "_" +
            str(row), str(year) + "_data_mask.shp")
    print (output_vecto)
    proc = subprocess.Popen(['C:\Program Files\GDAL\gdal_polygonize.py',
                             '-8',
                             '-f', 'ESRI Shapefile',
                             '-mask ' + data_mask_path,
                             data_mask_path,
                             output_vect_path],
                            stdout=subprocess.PIPE, shell=True)
    out, err = proc.communicate()

##############################################################################
##
# Reclassify map cubist output with everything below the detection threshold
# as 0 and everything equal to or above the detection threshold as 1
##############################################################################
##

DETECTION_THRESHOLD = 90

detected_disturbance_path = os.path.join(
    combined_training_data_output_directory,
    str(path) + "_" + str(row), str(year) + ".img")

detection_mask_path = os.path.join(
    combined_training_data_output_directory,
    str(path) + "_" + str(row), str(year) + "_detection_mask.img")

# Read in Raster
detected_disturbance_ds = gdal.Open(detected_disturbance_path, 0)

# Get in raster properties
driver = detected_disturbance_ds.GetDriver()
detected_disturbance_band = detected_disturbance_ds.GetRasterBand(1)
detected_disturbance_data = detected_disturbance_band.ReadAsArray(0, 0)

# reclass based on detection threshold
detected_disturbance_data[detected_disturbance_data<DETECTION_THRESHOLD] = 0
detected_disturbance_data[detected_disturbance_data>=DETECTION_THRESHOLD] = 1

# Mask NoData
detected_disturbance_data[data_mask_data == 0] = 255
data_mask_ds.GetRasterBand(1).SetNoDataValue(255)

# Write out new raster
detection_mask_ds = driver.CreateCopy(detection_mask_path,
```

```
                                        detected_disturbance_ds, 0)
detection_mask_ds.GetRasterBand(1).WriteArray(detected_disturbance_data)
detection_mask_ds = None

# Sieve data to remove noise
destination_path = destination_path = os.path.join(
    combined_training_data_output_directory,
    str(path) + "_" + str(row), str(year) + "_detection_mask-x8-11.img")

# 11 pixels ~ 1 hectare
proc = subprocess.Popen(['C:\Program Files\GDAL\gdal_sieve.py',
                         '-st', '11',
                         '-8',
                         detection_mask_path, '-of', 'HFA', destination_path
                         ],
                        stdout=subprocess.PIPE, shell=True)

out, err = proc.communicate()


##############################################################################
##
# Mosaic BAER data in path/row/year where the discover date is less than the
# last detection year scene date
##############################################################################
##

baer_dir = config.get("Directories", "BAER_DATA_DIRECTORY")
baer_dir = os.path.join(baer_dir, str(year))


scene_path = glob.glob(os.path.join(data_dir,
                                    '*' + str(year) + '????.img.gz'))[1]
scene_date = scene_path.split('_')[2][7:11]

scene_ds = gdal.Open('/vsigzip/' + scene_path, 0)
scene_geotransform = scene_ds.GetGeoTransform()
scene_upper_left_x = scene_geotransform[0]
scene_pixel_width = scene_geotransform[1]
scene_upper_left_y = scene_geotransform[3]
scene_pixel_height = scene_geotransform[5]
scene_xsize = scene_ds.RasterXSize
scene_ysize = scene_ds.RasterYSize

# loads a list of our rasters extent in the order [left,top,right,bottom]
extent = [
    scene_upper_left_x,
    scene_upper_left_y,
    scene_upper_left_x + (scene_xsize * scene_pixel_width),
    scene_upper_left_y + (scene_ysize * scene_pixel_height)
    ]

scene_geometry = ogr.Geometry(ogr.wkbPolygon)
ring = ogr.Geometry(ogr.wkbLinearRing)
ring.AddPoint(extent[0], extent[3])  # x1y1
ring.AddPoint(extent[2], extent[3])  # x2y2
ring.AddPoint(extent[2], extent[1])  # x2y1
```

```
ring.AddPoint(extent[0], extent[1])  # x1y2
ring.CloseRings()
scene_geometry.AddGeometry(ring)
ring = None

baer_paths = []
# Read in fire extents and see if they intersect the scene geometry
with open(os.path.join(baer_dir, 'perimeter.txt'), 'r') as f:
    for line in f:
        baer = line[:-1].split(',')
        baer_file_path = baer[0]
        baer_left = float(baer[1])
        baer_top = float(baer[2])
        baer_right = float(baer[3])
        baer_bottom = float(baer[4])
        baer_date = baer[5][4:]  # mmdd
        baer = None

        if baer_date <= scene_date:
            # Create polygon of the baer/disturbance extent
            baer_geometry = ogr.Geometry(ogr.wkbPolygon)
            ring = ogr.Geometry(ogr.wkbLinearRing)
            ring.AddPoint(baer_left, baer_bottom)  # x1y1
            ring.AddPoint(baer_right, baer_bottom)  # x2y1
            ring.AddPoint(baer_right, baer_top)  # x2y2
            ring.AddPoint(baer_left, baer_top)  # x1y2
            ring.CloseRings()
            baer_geometry.AddGeometry(ring)
            ring = None

            # If the baer/disturbance geometry intersects the scene
            # geometry then the scene likely has been disturbed so
            # extract disturbed pixels
            if scene_geometry.Intersects(baer_geometry):
                print baer_file_path
                baer_paths.append(baer_file_path)

output_baer_mosaic_path = os.path.join(
    combined_training_data_output_directory,
    str(path) + "_" + str(row), str(year) + "_baer_mosaic.img")

proc = subprocess.Popen(['C:\Program Files\GDAL\gdal_merge.py',
                         '-o', output_baer_mosaic_path,
                         '-pct',
                         '-ul_lr'] + [str(x) for x in extent] +
                        baer_paths,
                        stdout=subprocess.PIPE, shell=True)
out, err = proc.communicate()

# Mask BAER data with our Data Mask
output_baer_mosaic_masked_path = os.path.join(
     combined_training_data_output_directory,
     str(path) + "_" + str(row), str(year) + "_baer_mosaic_masked.img")

merged_baer_ds = gdal.Open(output_baer_mosaic_path, 0)
merged_baer_band = merged_baer_ds.GetRasterBand(1)
merged_baer_data = merged_baer_band.ReadAsArray(0, 0)
```

```
baer_mask = merged_baer_data * data_mask_data
baer_mask[baer_mask == 1] = 0
baer_mask[baer_mask == 2] = 0
baer_mask[data_mask_data == 0] = 255

output_ds = driver.CreateCopy(output_baer_mosaic_masked_path,
                              merged_baer_ds, 0)
output_ds.GetRasterBand(1).WriteArray(baer_mask)
output_ds.GetRasterBand(1).SetNoDataValue(255)
output_ds.GetRasterBand(1).FlushCache()

output_ds = None
```