Feature Extraction Techniques with Machine Learning for Satellite Image Classification

Zack Strathe
Kansas State University – CIS 731

Introduction

Satellite imagery has the potential to track a number of important indicators worldwide, such as deforestation, desertification, or crop health; however, with as much data as satellites are capable of capturing, there exists a challenge in gathering meaningful insights from such large collections of data. In particular, algorithms can be trained to analyze and classify satellite imagery, resulting in a trained model that could be used to detect and track environmental fluctuations, especially in remote regions of the world that may be otherwise overlooked. For instance, a trained satellite image classification model could use classification categories to track a specific region over time, and detect a change when the category changes.

With this project, my aim is to explore methods of feature extraction in images, which will be used to train a classification model. While a convolutional neural network is well-known to be state-of-the-art in this task, I am choosing to instead utilize a "classical" machine learning (ML) algorithm which allows more flexibility in feature extraction from images. My goal is to first determine an initial baseline— using no feature extraction (only the raw image pixel data)— by comparing ML classification algorithms, and then select the best-performing algorithms, based on the evaluation weighted-F1 score, for further experimentation with different feature extraction methods, with the end-goal of finding an extraction method that improves the classification weighted-F1 score over the baseline score.

Related Work

While many of the resources that come from a Google search of "image classification techniques" yield only information about image classification with a deep neural network, I managed to find a great web resource that goes into detail about methods of feature extraction from images, specifically for machine learning classification tasks (Ilango, 2017). This resource details methods of global feature extraction from images, such as quantifying color, texture, and shape. Another resource that I found provides information on modification of the pixel values as a method of extracting features (Singh, 2019). And another source (Brownlee, 2019) provides extensive information on feature selection techniques (such as reducing the number of features to only those that are statistically significant for prediction).

Data Set

The dataset for this project is a set of pre-tagged training and testing satellite images (specifically, images covering California) from Kaggle (Crawford, 2017). It contains 400,000 28x28 pixel 4-band (red, green, blue, and near-infrared) training images and labels, and 100,000 testing images and labels, and is roughly 7 GB in size. The data labels categorize the images as either "barren land", "trees", "grassland", or a fourth "other" category for images that don't fit the first three. Conveniently, the image testing and training sets are already encoded into '.csv' files. In X_train and X_test, each line is a list containing the pixel-by-pixel color value of each image, with each pixel represented by the 0 to 255 color value in red, green, blue, and near-infrared respectively. In y_train and y_test, the labels supplied with the dataset are one-hot-encoded and take the format of [1,0,0,0] for the "barren land" label, [0,1,0,0] for the "trees" label, [0,0,1,0] for the "grassland" label, and [0,0,0,1] for the "other" label. The dataset is noted on Kaggle to have been hand-labeled by using a "labeling tool developed as part of this study." Initially, the images were sourced from tiles that were approximately 6,000 pixels wide by 7,000 pixels high, which were manually labeled. Next, the labeled images were split into 28x28 pixel samples by using "non-overlapping sliding window blocks." Unfortunately, more details about the initial labeling process are not available, and I will assume these labels to be ground truth for analysis.

Methodology

Platform

I'm using PySpark exclusively to load the data, pre-process the data, train a classifier model, and evaluate the model. I selected Google Colab to develop this project with, because it is cloud-based and can be easily switched from a free instance to a hosted instance from Google Cloud, when more dedicated computing resources are needed. Af-

ter considerable frustration and struggling with a free instance of Google Colab to load and pre-process the full 500,000 image and label dataset, I decided to utilize a hosted VM (with v16 CPU and 64 GB memory) from Google Cloud. In addition, I did some limited testing with an Amazon Web Services EMR notebook, hosted on a small 3-VM cluster, with the data loaded from an AWS S3 bucket. And while the AWS EMR cluster performed better than a single-VM notebook, the cost was considerably higher and therefore I chose to forego using it for the implementation of this project. When utilizing a single-machine for running PySpark with this dataset, it's important that the machine has plenty of memory (because some files are multiple GB in size and PySpark RDD operations will create new RDDs in memory). Further, it's necessary to initialize PySpark with a configuration that takes advantage of all available memory and processing capacity. Accordingly, prior to initializing a SparkContext, I used the SparkConf class to set custom configuration parameters.

Data Preprocessing

To initially read the data, I utilized the PySpark sqlContext.read.csv() function to import each .csv file (X_train, Y_train, X_test, and Y_test) as a dataframe. For ease of data manipulation, I decided to use the MLlib RDD-based functions. A RDD (Resilient Distributed Dataset) in PySpark is a collection of immutable data that can be operated on in parallel and lacks a schema, which is what makes it different from a dataframe (DataBricks). And because the majority of operations for this project will be only on a single column data where there is no need for a schema, it makes sense to utilize RDDs. Once each of the data files has been converted into an RDD, no further preprocessing is necessary before transforming labels and extracting features from the data.

For this project, I utilized a number of functions for the data transformation operations. One main function (data_preprocess_main()) serves as the first step of initializing each of four RDDs, and another function (data_rdd_process()) serves to execute mapping operations to RDDs. And further functions are defined for specific RDD mapping operations for which detail follows.

The labels supplied with the dataset are one-hot-encoded, but the MLlib RDD-based algorithms require training data to be formatted as a LabeledPoint, which further requires the labels to be formatted as a number (Apache). Therefore, the labels (Y_train and Y_test) need to be converted by mapping each RDD to a function (convert_label()) that will return the label converted to a numerical data type (I used a float but an integer would likely work as well).

For testing different feature extraction methods, the original rows in each of X_train and X_test need to be mapped over with functions that will return each row transformed. In this project, I implemented two functions that accomplish this task in different ways: pixels_transform(), which modifies the image data by utilizing NumPy arrays, and cv2_transforms(), which transforms the original image data with functions from OpenCV.

For transformations with NumPy arrays, there are 3 methods of feature extraction that I evaluated. For all methods, the image data was first converted from a row of 3,136 integers into a 28x28x4 array. The first transformation (flatten_pixels) gets the mean value for each pixel (ignoring the 4th value that represents the near-infrared spectrum value). The second transformation (infra_only) gets and returns only the near-infrared value for each pixel. And the third transformation gets the pixel mean, and appends the near-infrared value to it. Each of these transformations, with the resulting number of features for each image is shown below in Figure 1.

Parameter	Transformation Effect	Resulting Number of Features per Image
none (default)	n/a	3,136
flatten_pixels	returns the mean of the RGB values for each pixel	784
infra_only	returns only the infrared value for each pixel	784
flatten_plus_infra	returns the mean of the RGB values, and the infrared value for each pixel	1,568
edges_only	returns an array of edges detected using cv2.Sobel	784
edges_plus_pixels	returns an array of edges, collated within each pixel sub-array in the default image data	3,920
hu_moments	returns an array of HuMoments calculated from cv2.HuMoments	7
hu_moments_plus_pixels	returns an array of HuMoments, appended to the end of the default image data array	3,143
histogram_greyscale	returns a greyscale histogram array, binned by RGB value (0-255)	256
histogram_greyscale_plus_pixels	returns a greyscale histogram array, binned by RGB value (0-255), appended to the end of the default image array	3,392

Figure 1: Feature extraction parameters with description of the transformation effect and resulting number of features per image

For transformations with OpenCV, there are 6 methods of feature extraction that I evaluated. For all of these transformations, the image data was first converted into a 28x28x3 array (ignoring the near-infrared values for each pixel), and then converted to grayscale. The first transformation (edges_only) utilizes the cv2.Sobel edge detection algorithm to find edges in the image, which are then flattened into a single list and returned. The Sobel algorithm detects edges by finding where there are sudden changes in pixel intensity, using a 3x1 kernel (Mallick). The second transformation collates the detected edges into the original list. Since the location of the edges are important, I made the assumption that collation would return better results than simply appending the list of edges onto the end of the original list. The third transformation (hu_moments) utilizes the cv2.HuMoments to return a list of 7 numbers meant

to characterize the shape of objects in an image (Hu, 1962). The fourth transformation (hu_moments_plus_pixels), returns the HuMoments appended to the end of the original image data. The fifth transformation (histogram_greyscale), returns the greyscale image converted into a histogram (separated into 256 bins, to represent each greyscale color value). And the sixth transformation (histogram_greyscale_plus_pixels) returns the greyscale histogram appended to the end of the original image data list. All of these transformations are shown with the corresponding number of features in Figure 1.

After transformations have been completed on both X_train and Y_train, both of the resulting RDDs need to be joined, which I discovered can potentially be tricky with PySpark. Because the original data files do not contain an index, both RDDs need to be mapped over to add one. PySpark contains a useful function for this purpose called zipWithIndex(), which will add an index to the end of each RDD row (though I question why it works that way, because, for a join to work properly, I had to map each RDD again, to re-order each row with the index first and data item second). Another issue that I had with joins in PySpark was that the resulting RDDs from a join will not be ordered. Though fortunately this issue with joins had no effect on model training or evaluation results, and is easily solved by using the PySpark sortBy() function immediately following a join. Following joining X_train and Y_train into a single RDD, the resulting RDD needed to be converted into a list of LabeledPoints that consist of a numerical label and data formatted as the MLlib RDD-based Vectors class.

For the final evaluation step, I attempted to further refine the best-performing method of feature extraction with two feature selection methods. First, the ChiSqSelector from MLlib was used to test whether the features could be further refined by selecting the top number of features. The ChiSqSelector uses a chi-squared test to determine which features to keep, according to a specified limit in the top number of features. Second, the MLlib Normalizer was used to test whether normalizing the features improved the evaluation score.

Evaluation

To simplify evaluation of algorithms and feature selection methods, I implemented a function for model evaluation that takes the testing and training RDDs as inputs and outputs the evaluation metrics. Doing this allowed me iteratively evaluate the models without considerable difficulty.

Following selection of the best feature extraction method, along with the corresponding classification algorithm and feature selection method, I conducted a 10-fold cross-validation evaluation of the resulting model to verify results. For each fold of testing, I used a union to combine X_train and X_test together into a single RDD, then did the same with Y_train and Y_test. Next, I joined both of the combined RDDs into a single RDD, consisting of all 500,000 images and labels. To accomplish 10-fold cross-validation testing, I used the rdd.randomSplit() function to split the RDD into sets consisting of 80% training, and 20% testing. In addition, I passed a seed value to the randomSplit() function to ensure that each fold would be split uniquely. So, by iterating through a range of 10 unique integers, the data was split uniquely for each evaluation fold. Each fold of the model was then evaluated using the evaluation function to get the weighted-F1 score. To statistically confirm whether the null hypothesis (baseline model) could be rejected, both the baseline model and improved model were trained and evaluated with identical data for 10 folds. Each model was pre-processed, then each split using the same seed before being evaluated. And with each fold of evaluation results for each model being from the exact same set of training and testing data, I was able to compute a paired t-test statistic and p-value from the weighted-F1 scores, using the ttest_rel() function from SciPy.

Visualization

Following cross-validation of the best model, I created an output file to use for visualizations of the resulting classifications. This was done so that I could load the file at a later date without needing to process the data again. With the output file, I utilized matplotlib to generate a subplot figure displaying the images, along with predicted and actual labels, and alongside a graph plot of the features extracted from the image.

Models

For initial selection of classification algorithms, I evaluated six classification algorithms from the Spark MLlib library using the unmodified image data. The algorithms evaluated were Random Forest, Decision Tree, Logistic Regression, Naïve Bayes, Gradient Boosted Trees, and Support Vector Machine. For each evaluation, the model was trained on the full training set of 400,000 images and labels, then evaluated with the MLlib MulticlassClassificationEvaluator with the full testing set of 100,000 images and labels. The results are displayed in Figure 2, which is sorted descending by weighted-F1 score (which is established as the evaluation metric under "Evaluation Metrics" below).

Algorithm Name	Precision Score	Recall Score	F1 Score	Accuracy Score	Total Time
Random Forest	0.82	0.81	0.81	0.81	287.75
Decision Tree	0.77	0.76	0.76	0.76	286.21
Logistic Regression	0.73	0.74	0.73	0.74	3,373.62
Naive Bayes	0.57	0.51	0.51	0.51	231.45
Gradient Boosted Trees	0.31	0.44	0.33	0.44	1,465.30
Support Vector Machine	0.04	0.20	0.07	0.20	247.38

Figure 2: Classifier algorithm evaluation results, with unmodified image data

Looking at the table, it is clear that the top 3 (Random Forest, Decision Tree, and Logistic Regression) performed fairly well at classification using the unmodified image data. And since the Random Forest model performed the best, with a weighted-F1 score of 0.81, it will be the base-

line to compare with results of different feature extraction methods.

For testing several different feature extraction methods, I'm utilizing Random Forest, Decision Tree, and Logistic Regression algorithms. Rather than evaluating with just one algorithm, this makes it more likely that the ideal model is found, as different types of features extracted from the images may drastically alter how an algorithm performs. And while Logistic Regression took substantially longer than the other two classification algorithms in the initial evaluation, I am including it with the assumption that the total time is negligible, because excessive processing time would be avoided if this model were deployed to an actual Spark cluster that could be scaled out with additional "workers" as-needed.

Evaluation of Feature Extraction Methods

Baseline

As already established with the initial testing of MLlib RDD-based algorithms, the baseline for this project is an evaluation weighted-F1 score of 0.81, which was set by the Random Forest classifier model using the unmodified image data. As such, the null-hypothesis is that no feature-extraction methods will improve the weighted-F1 score, and the alternative-hypothesis is whether each method (evaluated with each of Random Forest, Decision Tree, and Logistic Regression classification algorithms) returns a weighted-F1 score greater than 0.81. And further, if any methods return a higher weighted-F1 score, I will verify that the null-hypothesis can be rejected by computing a paired t-test statistic from 10-fold cross-validation results between the baseline and improved model.

Evaluation Metrics

To evaluate classification models, accuracy, which is the ratio of correct predictions to total predictions, seems like the ideal metric to use as a comparison. However, because this is a multiclass classification problem, and there is no guarantee that the model will have an equal number of training samples from each classification category (meaning that the model may have some classification bias), it's better to utilize weighted precision and recall metrics. Precision is the ratio of true positives to total positives, while recall is the ratio of true positives to the sum of true positives and false negatives (Google). And since I want a model that maximizes both precision and recall for each category, I chose to use the weighted-F1 score, which is the harmonic mean of precision and recall, to rank models. The weighted-F1 score will be calculated with the MLlib MulticlassClassificationEvaluator.

Results

For each feature extraction method, the training image data was transformed using that method, then trained with Random Forest, Decision Tree, and Logistic Regression algorithms. Each resulting model was then evaluated with the testing image data (which was also accordingly transformed). The results are shown in Figure 3, sorted in descending order by weighted-F1 Score.

Algorithm Name	Feature Extraction Method	Precision Score	Recall Score	F1 Score	Accuracy Score	Total Time
Logistic Regression	*histogram_greyscale*	0.93	0.92	0.93	0.92	591.54
Logistic Regression	*histogram_greyscale_plus_pixels*	0.92	0.92	0.92	0.92	4,348.75
Logistic Regression	*edges_plus_pixels*	0.88	0.88	0.88	0.88	5,217.39
Decision Tree	*histogram_greyscale_plus_pixels*	0.88	0.87	0.88	0.87	543.25
Decision Tree	*histogram_greyscale*	0.85	0.86	0.85	0.86	367.05
Random Forest	*histogram_greyscale_plus_pixels*	0.85	0.83	0.83	0.83	580.61
Random Forest	*hu_moments_plus_pixels*	0.83	0.83	0.82	0.83	771.57
Random Forest	*histogram_greyscale*	0.83	0.83	0.82	0.83	384.29
Random Forest	*edges_plus_pixels*	0.83	0.82	0.82	0.82	861.28
Random Forest	*flatten_plus_infra*	0.79	0.79	0.78	0.79	677.44
Decision Tree	*edges_plus_pixels*	0.77	0.76	0.76	0.76	798.13
Logistic Regression	*hu_moments_plus_pixels*	0.76	0.76	0.76	0.76	4,267.64
Decision Tree	*hu_moments_plus_pixels*	0.77	0.77	0.76	0.77	744.53
Decision Tree	*flatten_plus_infra*	0.73	0.73	0.73	0.73	653.72
Decision Tree	*hu_moments*	0.73	0.73	0.71	0.73	385.65
Random Forest	*hu_moments*	0.73	0.72	0.71	0.72	392.47
Decision Tree	*flatten_pixels*	0.67	0.68	0.67	0.68	471.40
Random Forest	*flatten_pixels*	0.58	0.69	0.62	0.69	489.09
Random Forest	*infra_only*	0.55	0.61	0.53	0.61	296.88
Decision Tree	*infra_only*	0.49	0.59	0.52	0.59	293.86
Logistic Regression	*flatten_plus_infra*	0.46	0.46	0.46	0.46	2,357.31
Decision Tree	*edges_only*	0.38	0.44	0.38	0.44	388.65
Logistic Regression	*hu_moments*	0.47	0.44	0.37	0.44	402.00
Random Forest	*edges_only*	0.29	0.46	0.35	0.46	401.36
Logistic Regression	*flatten_pixels*	0.33	0.30	0.23	0.30	1,382.87
Logistic Regression	*edges_only*	0.14	0.35	0.19	0.35	1,147.40
Logistic Regression	*infra_only*	0.15	0.28	0.17	0.28	1,193.51

Figure 3: Evaluation results of feature extraction methods with Random Forest, Decision Tree, and Logistic Regression classification algorithms.

Based on the evaluation results, the "histogram_greyscale" method of feature extraction, with the Logistic Regression algorithm, was the best performer, with a weighted-F1 score of 0.92. An interesting observation from these results is that, while this feature extraction method significantly improved results with the Logistic Regression algorithm, the other algorithms tested with this method fared only slightly better or worse than the original evaluations with unmodified image data. With the best feature extraction method identified (of the limited number tested), I attempted to further improve the set of features by utilizing feature selection methods available in MLlib. A table of results detailing the feature selection methods, and new evaluation scores, is shown in Figure 4.

New Parameter	Chi-Sq Num	Algorithm Name	Feature Extraction Method	Precision Score	Recall Score	F1 Score	Accuracy Score	Total Time
Normalize Features	*N/A*	*Logistic Regression*	*histogram_greyscale*	0.93	0.93	0.93	0.93	609.16
Chi-Sq Selection	*200*	*Logistic Regression*	*histogram_greyscale*	0.92	0.91	0.91	0.91	779.75
Chi-Sq Selection	*100*	*Logistic Regression*	*histogram_greyscale*	0.73	0.75	0.73	0.75	692.11

Figure 4: Evaluation results of 'histogram_greyscale' feature extraction method and Logistic Regression algorithm, with additional feature selection methods, sorted descending by weighted-F1 Score

Based on comparing between the tables in Figure 4 and Figure 3, normalizing the features did improve the precision and accuracy scores for the best model by 0.01, but did not significantly boost the weighted-F1 score. However, this is still an improvement, and did not add significant computation time. On the other hand, the chi-square selection method caused evaluation results to worsen.

After determining the best-performing model to be with "histogram_greyscale" feature extraction, with normalized features, and with the Logistic Regression algorithm, I computed 10-fold cross-validation results for both the baseline and improved model, then used those results to compute a paired t-test statistic and p-value. The mean result from cross-validation slightly improved the original baseline weighted-F1 score (from 0.81 to 0.82), and evaluation score of the improved best-model decreased slightly, and achieved a mean weighted-F1 score from cross-validation of 0.92. Using cross-validation testing results, the t-test statistic between the baseline and improved model was calculated with a statistic of -236.343 and p-value of 2.21E-18. Since the p-value is less than 0.05, I conclude with 95% confidence that there is a statistically significant improvement versus the baseline model.

Conclusion

Of the feature extraction methods tested, the method of generating a greyscale histogram from the image, along with utilizing the Logistic Regression algorithm and normalizing the features, returned the best results, improving the baseline weighted-F1 score from 0.81 to 0.93. A visualization, displaying a manually-selected sample of images, with a correctly and incorrectly predicted image for each class, is shown in Figure 5. This visualization also displays the greyscale histogram that was extracted from each image.

Future Work

Unfortunately, time-restrictions limited the number of potential feature extraction methods that I was able to implement and evaluate. While the 0.93 weighted-F1 score that the best model achieved was an improvement in the baseline, I believe that there is still room for improvement beyond that result. Due to the success of using greyscale histogram features in improving image classification accuracy, I believe that there is potential for further tests with alternate methods of generating histograms from images, such as using the full RGB or near-infrared pixel values instead; or alternatively, concatenating the histogram with other global image extraction features, such as Hu Moments. In addition, limiting my evaluation to the MLlib RDD-based algorithms did omit some algorithm options that are included only with the MLlib dataframe-based library, such as a Multilayer Perceptron Classifier, a One-Vs-Rest classifier, and a Factorization Machines classifier. I believe that there would be benefit from further testing to determine how those algorithms perform.

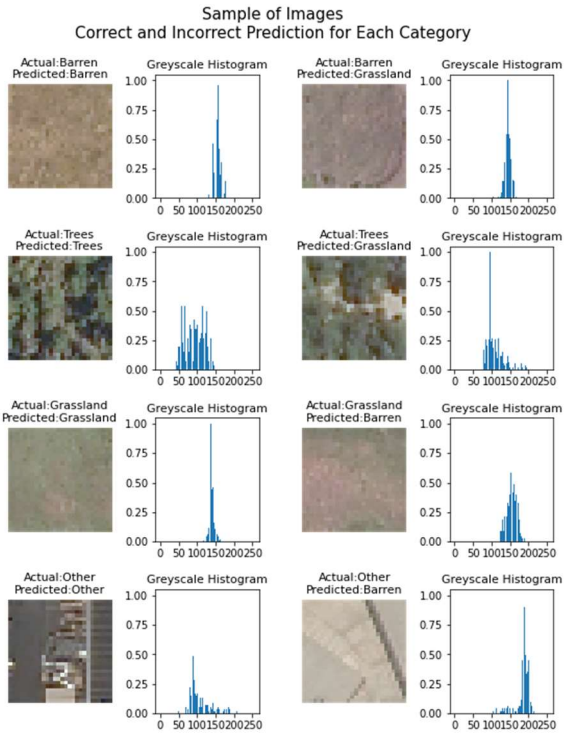

Figure 5: A sample of images corresponding to each classification category (one correct and incorrect classification for each), as well as the extracted greyscale histogram

An important consideration is whether the models developed for this project would actually perform well with unlabeled satellite imagery to accomplish monitoring tasks similar to those described in the introduction. Unfortunately, any real-world benefit of the model would be limited, due to there being far more than 4 basic-categories that satellite imagery could fall under. Therefore, there would be benefit for future work to add many more classification categories, such as "mountainous", "water", or "clouds" (to identify regions where clouds had obscured the image, and should therefore have new imagery sourced) for examples.

References

Apache. Apache Spark Documentation. MLlib: RDD-based API Guide. https://spark.apache.org/docs/latest/mllib-guide.html.

Brownlee, Jason. 2019. How to Choose a Feature Selection Method For Machine Learning. Machine Learning Mastery. November 27, 2019. https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/.

Crawford, Chris. 2017. DeepSat (SAT-4) Airborne Dataset. Kaggle. 2017. https://www.kaggle.com/crawford/deepsat-sat4.

DataBricks. DataBricks.com. Resilient Distributed Dataset. https://databricks.com/glossary/what-is-rdd.

Google. Machine Learning Crash Course. Google. https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall.

Hu, Ming-Kuei. 1962. Visual pattern recognition by moment invariants. 1962, Vol. vol. 8, no. 2, pp. 179-187.

Ilango, Gogul. 2017. Image Classification using Python and Scikit-learn. 2017. https://gogul.dev/software/image-classification-python.

Mallick, Satya. LearnOpenCV. Edge Detection Using OpenCV. https://learnopencv.com/edge-detection-using-opencv/.

Singh, Aishwarya. 2019. 3 Beginner-Friendly Techniques to Extract Features from Image Data using Python. Analytics Vidhya. August 29, 2019. https://www.analyticsvidhya.com/blog/2019/08/3-techniques-extract-features-from-image-data-machine-learning-python/.