Seeing the Forest for the Trees: Utilizing Random Forest Machine Learning to Classify Forest Coverage

Peter McKay and Daniel Walinsky
Department of Computer Science and Information Science, University of Oregon
Eugene, Oregon
{pem,dwalinsky}@cs.uoregon.edu

ABSTRACT

In this paper, we consider the problem of automatic forest coverage classification, as posed by the Kaggle competition: "Forest Cover Type Prediction." In short, we use python's machine learning libraries to construct a model capable of reliably predicting the predominant type of tree cover for a small region of forest, given a collection of cartographic variables. We create such a model by leveraging a voting body of a number of different classifiers, with a second layer of classifiers for certain cases. We optimize this technique by carrying out a number of feature transformations and subsequent feature scaling operations, making use of a number of statistical relationships discovered through judicious application of graphical data analysis tools.

We find that the classifier thus developed is capable of reliably predicting accurate forest cover with approximately 80% accuracy. In reviewing the particulars of the performance of our algorithms, we gain some insight into the data set, and the problem domain in general.

1. INTRODUCTION

Education sometimes seems like a process of ever-increasing specialization. Preschool acclimatizes larval humans to basic scholarly etiquette (listening, sharing, naptime) grade-school begins to separate learning into a number of different subjects, and by the time college rolls around one is expected to pick a major. By the time one has reached the PhD program, one's course of study has narrowed down to a small tunnel of the unknown, sometimes just a single problem.

When dealing with questions on the frontiers of science, it is not uncommon to encounter a problem that falls within the purview of many such narrow fields, necessitating the addition of more and more team members. This is something a problem, because, as any software engineer can tell you [**?**], throwing more people at a problem is not exactly a solution. Despite how interested our University Administration seems in the subject, interdisciplinary research just doesn't seem to scale very well.

One of the real draws for machine learning lies in allowing us to compartmentalize our need for understanding, avoiding issues by transforming our problems into a familiar domain and extracting the answers we need. So long as our model "understands" a problem, we can ignore a certain level of complexity and move on to the solving the interest parts of the problem. This allows a computer scientist with an aversion to the so-called "outdoors" to make a number of correct judgments about forests, all from the comfort of their office.

From our limited experiences in the outdoors, we can make a few judgments about the construct of a forest given a few data points. An accomplished expert in the fields of ecology or geography, specializing in the climate of that forest, might be able to tell us quite a bit more information based on that same starting data. In the noble tradition of computer science, we would like to avoid doing that much work, and we will apply machine learning techniques to accomplish this task. To that end, we consider the following multiclass classification problem. We construct a classifier that, when presented with a number of cartographic variables that apply to a 30–30 meter cell of forest, classifies such a cell by the predominant type of tree cover found therein.

In this paper, we utilize a multi-level ensemble method of classification. We add a number of new features, scale the feature vector appropriately, and then train a collection of different classifiers on that data. The classifiers vote to determine the classification of an example, and in some cases, pass on that vote into another layer of machines trained on a smaller subset of the training data. We accomplish these tasks using Python and a series of libraries designed for data analysis and machine learning. In so doing, our model reaches approximately 80% accuracy on the Kaggle data set.

Elevation	Elevation in meters
Aspect	Aspect in degrees azimuth
Slope	Slope in degrees
Horizontal_Distance_To_Hydrology	Meters to nearest water-body
Vertical_Distance_To_Hydrology	Meters to nearest water-body
Horizontal_Distance_To_Roadways	Meters to nearest roadway
Hillshade_9am (0 to 255 index)	Hillshade index at 9am
Hillshade_Noon (0 to 255 index)	Hillshade index at noon
Hillshade_3pm (0 to 255 index)	Hillshade index at 3pm
Horizontal_Distance_To_Fire_Points	Meters, wildfire ignition points
Wilderness_Area	Wilderness area designation
Soil_Type	Soil Type designation

Table 1: Feature List

2. BACKGROUND

2.1 Problem Structure

The forest coverage classification problem clearly fits into the category of a multiclass classification problem, in which we describe a classifier that maps from a feature vector to one of a series of possible output labels. The output labels are given below:

1. Spruce/Fir
2. Lodgepole Pine
3. Ponderosa Pine
4. Cottonwood/Willow
5. Aspen
6. Douglas-fir
7. Krummholz

Unlike a binary classifier, a multiclass classifier can quite easily reach error rates in excess of 50%. As we are dealing with data collected from a natural source, we expect to encounter a fair amount of noise within the data set. We keep this in mind when selecting the classifiers enumerated in the following subsection, and when tuning hyperparameters which govern noise tolerance.

The feature vector provided for forest cover type classification is outlined in table 1. It contains a mix of continuous and discrete binary values.

The continuous features are graphed in figure 1, and include elevation, slope and aspect, as well as distances to important geographic points, such as water and roads. Examining these graphs, we conjecture that elevation may be one of the more important features: it looks like some cover types are almost entirely constrained to a small range of elevation. While the other continuous features may be helpful later on, their importance is not immediately visible from a simple graph.

The discrete features, as visualized in figures 2 and 3, account for multiclass features encoded with the one-hot encoding method. It is clear from these graphs that some cover types are more prevalent in certain wilderness areas or certain soil types. However, determining the forest cover by wilderness area or soil type alone proves impossible, as no wilderness area or soil type is the sole domain of any type of forest cover. Therefore, we must integrate the discrete features into our model alongside the continuous features in order to produce an effective algorithm.

2.2 Solution structure

The classification solution we assemble to solve this problem involves a number of different machine learning models. We began the project by attempting a Random Forest classifier, but eventually expanded to also make use of a Gradient Boost Machine, a Support Vector Machine, and a k-Nearest Neighbor classifier. Each of these machines has its own strengths and weaknesses, which we will discuss as follows. Each model has a number of hyperparameters that must be tuned properly before evaluating the total efficacy of such a technique: we will cover the tuning process in the following sections.

A Random Forest classifier [**?**] is an example of an ensemble method, a sort of mashup of decision trees with a machine learning technique called bootstrap aggregating, or, more colloquially, bagging [**?**]. Instead of constructing a single tree on the entirety of the training set, Random Forest constructs many smaller trees on randomly selected subsets of the training data. This has a number of perks: Random Forests are resistant to artifacts of training set ordering and are highly robust in the face of noise and superfluous features. Perhaps most importantly, Random Forests, by effectively averaging together a number of decision trees, can effectively mitigate the tendency of decision trees to overfit the training data. As a small bonus, the structure of the algorithm allows for efficient parallelism, making this a very quick algorithm on a computer with sufficient cores.

When we began exploring the idea of adding further classifiers, we settled on the Grading Boosting Machine [**?**] after some small amount of testing. Like Random Forest, GBM is an ensemble method that constructs a number of subsidiary decision trees. In this case, the algorithm combines decision trees with the idea of boosting [**?**]. Boosting is, in abstract, the process of combining a number of poor predictors into a single good predictor. In the case of a tree-based GBM, our process is superficially similar to bagging, insofar as we create trees based on repeatedly selecting subsets from the total training set. The difference lies primarily in how we weight such trees. Instead of relying largely on the sheer mass of trees, we instead attempt to select subtrees that will improve currently weak areas of the aggregate predictor.

k-Nearest Neighbor [**?**] is arguably the simplest algorithm we deploy in the course of this paper. We consider an n-dimensional graph (for a total of n features) over the feature space. By populating the graph with our training examples, our predictor takes the form of simply checking the class of the k training examples that are closest to the testing instance. This distance metric can be a simple Euclidean distance calculation, or one of a number of more complex metrics suited for particular problem domains.

Support Vector Machines [**?**] share some commonalities with the k-NN algorithm as described above. We repeat the process of populating an n-dimensional graph with training examples, and from there the algorithms diverge significantly. k-NN is a lazy algorithm: that is to say, computation is generally delayed until it must actually compute the classification. By contrast, an SVM is a non-lazy binary linear classifier. The SVM attempts to generate a hyperplane that separates the graph into two distinct sets. By utilizing the approach of one-hot encoding, we can cause SVM to carry out multiclass classification. Our SVM model can be made resistant to overfitting, and SVMs in general can make use of the kernel trick to classify nonlinear functions. In this case, we make use of the RBF [**?**] kernel, which tends to be useful in smoothing out noisy data sets.

Most of the algorithms we made use of for this classification problem were provided by scikit-learn, with a couple of important exceptions. When mixing our classifiers together into a voting body, we rolled our own method of voting. In addition, the confusion matrix as detailed in figure 4 motivated a second layer of Random Forest classifier, where certain classes of classifications cause our system to double-check its work.

3. METHODOLOGY

3.1 Feature engineering

The first step in any machine learning problem is to take a look at the data, and see what we've gotten ourselves into. If we start by looking at the continuous variables, we observe a number of interesting correlations.

Aspect appears to be somewhat of an outlier in its shape, which makes sense when one considers the variable is given in terms of degrees: we should be working in modulo 360. There are a number of transformations that could solve this issue: we choose to simply split aspect into two different features scaled to 180°.

We know that Elevation is measured in meters, as are the 5 other

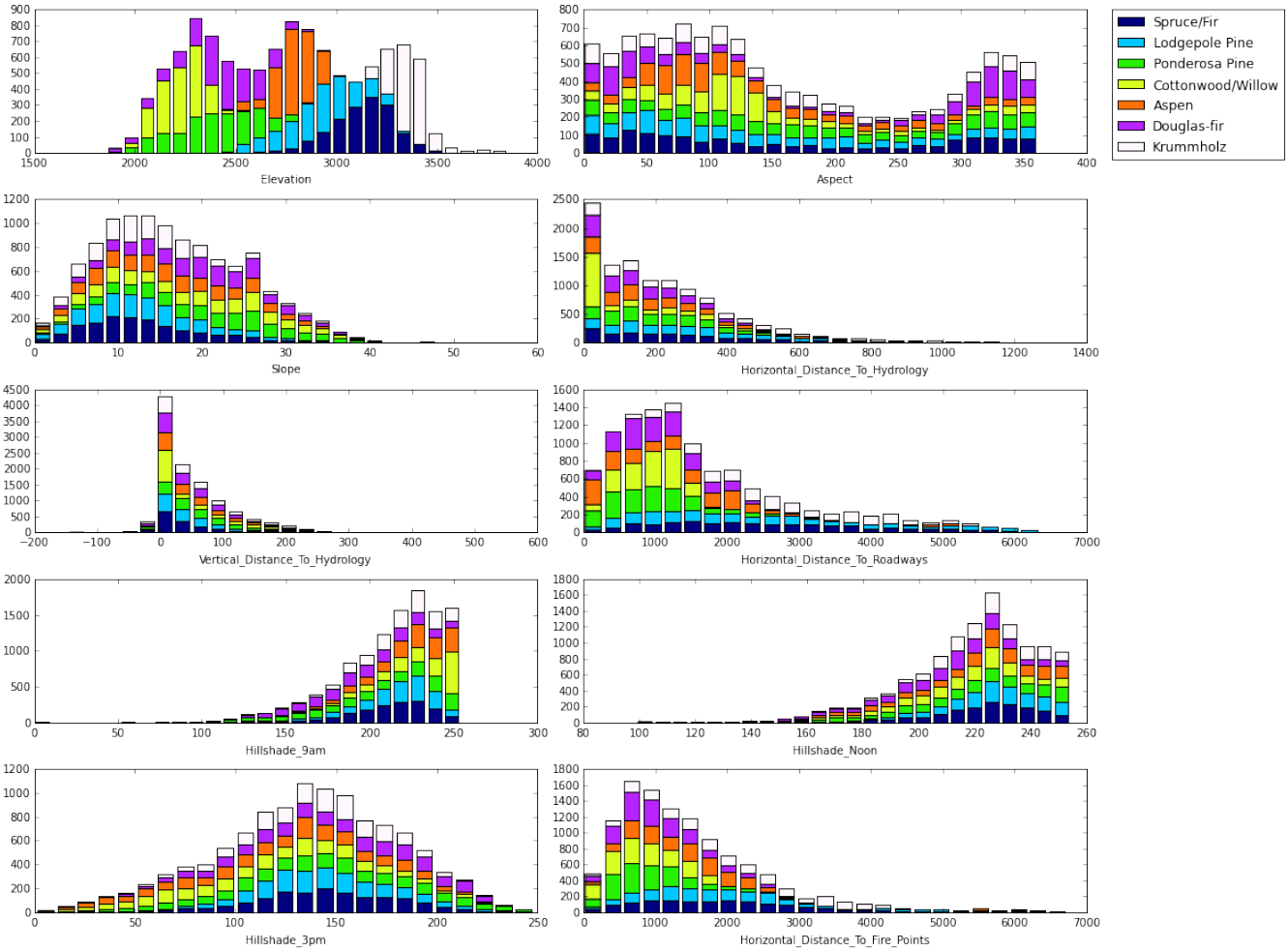

Figure 1: Histogram for continuous variables showing cover type distribution

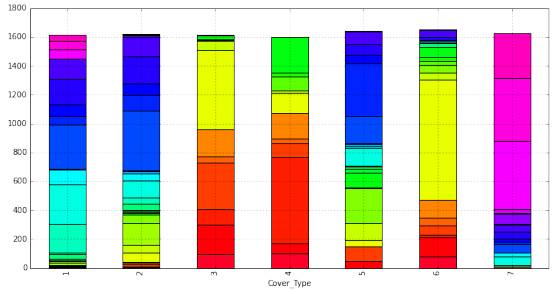

Figure 2: prevalence of soil type aggregated by cover type

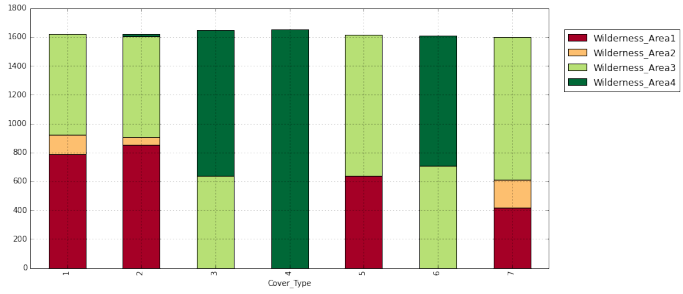

Figure 3: Wilderness areas grouped by cover type

"distance to nearest item of interest" features. It seems like we should be able to do something with that. In our case, we add the differences between Elevation and the Hydrology Distance features. We also integrate the horizontal and vertical distance features, for a Euclidian distance measurement. We notice there are some forest cells that appear to be nearly underwater, according to the Vertical Distance To Hydrology feature. We introduce a new boolean feature to denote that possibility.

While the graph of wilderness areas appears initially promising (look at all of those solid blocks of color!), we note that this is an artifact of the graphing process: there are only a few wilderness area types, and they seem to be spread pretty evenly among a the cover types.

The soil type graph shows us a preview of one of our more interesting discoveries, which we cover more thoroughly in the next subsection. For now, we simply observe that some cover types are more distinct from the group than others, when we consider solely the soil types.

In order to optimize the performance of the SVM and k-NN, we utilize the scikit standard scaler to scale our continuous variables to uniform maximums. This scales each feature to have an average value of 0 and a standard deviation of 1. SVMs is known to perform better with scaled data, and we observed a similar increaes in the validation accuracy of k-NN as well.

3.2 final model

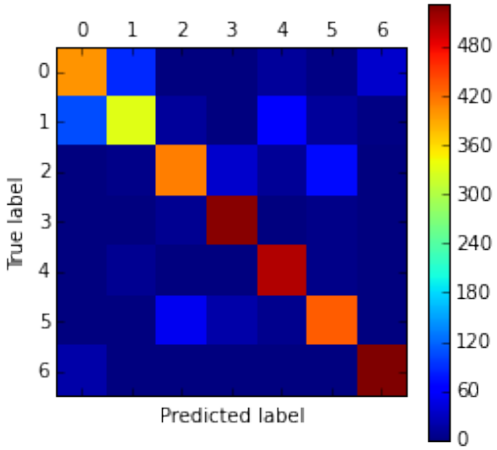

Figure 4: Confusion Matrix

As we add new features and massage existing features into new shapes, we iteratively apply our classifier scheme to the feature vector to test the efficacy of those features. This, however, is not an exact science. A change that increases accuracy over some decisions could very well lead to a loss of accuracy over other decisions. In order to more fully examine the consequences of our actions, we make use of a graphical tool called a confusion matrix [**?**]. A confusion matrix, or, error matrix, is a tool suited for specifying exactly what kind of misclassifications lead to a loss of accuracy. In our case, as we see in Figure 4, our model does a pretty good job until we start trying to differentiate between the Spruce/Fir type and the Lodgepole Pine type (and, to a lesser extent, between Lodgepole Pine and Douglas-fir).

Adding a mechanism to our ensemble to correct for this error results in a modest accuracy increase for our validation set, but a significantly greater accuracy gain for the Kaggle competition itself. If we count up the estimated classifications, we observe that our classifiers predict more 1s and 2s than any other cover type, by nearly an order of magnitude. This suggests that the unexpected gain in accuracy is due to a heavier preponderance of Spruce/Fir and Lodgepole Pine trees in the testing region.

Informed by these features, we then tested a number of classifiers with default settings. Some of these classifiers performed very poorly, such as the Naive Bayes algorithm. We selected several of the best performing algorithms, and collected them into a voting body. We generated a Support Vector Machine, a Gradient Boosting Machine, and a Random Forest classifier on the training data. We use the weights from the Random Forest only to optimize the performance of a k-Nearest Neighbor classifier. That done, we discard the first Random Forest weight vector, and train a new Random Forest classifier on a subset of training data composed only of cover types 1 and 2.

The GBM, SVM, k-NN classifier, thus assembled, use majority voting to determine which classification is selected for a given collection of features. If such a vote results in a classification of cover type 1 or 2, we fall back to our more select 1/2 Random Forest classifier.

4. RESULTS

In order to approximate maximum accuracy values for our models, we carried out a series of tuning operations by way of an iterative binary search. One hyperparameter at a time, we preceded to run the classifier with converging values until we had reached a sufficiently small difference in accuracy. We recorded these values and explored the impact that small changes in multiple hyperparameters had on the accuracy of the model.

The first classifier we attempted was Random Forest, primarily out of an appreciation for the pun value of using a forest to classify forests. Initially, our accuracy hovered between 60% and 70% on our validation data. When we tuned the hyperparameters (increasing the number of features considered when splitting a tree, adding more trees to the forest, etc) of the Random Forest model, we reached 80% accuracy on our validation set. Unfortunately, our accuracy on the Kaggle competition remained below 70%.

We undertook similar experiments with the Gradient Boosting classifier (optimal results with an increased learning rate of approximately one third, increased max depth of five, increasing the number of boosting stages to 400, examining all features for a split), with somewhat better results (peak accuracy of 85% on the validation set).

We found that a k-value of 5 and a Euclidean distance measure produced the highest-quality classifier from the k-Nearest Neighbor, model. Weighting the input features with values produced by a Random Forest significantly improved the accuracy on validation data. Surprisingly, renormalizing the weighted features slightly gave an even higher accuracy over simply applying the weights to each feature. Under these settings, we reached a validation accuracy of about 82%.

Using an RBF kernel and a surprisingly high penalty parameter, with arbitrarily many iterations, our SVM reached approximately 81% accuracy.

We attempted a Naive Bayes model, but were unable to reach an accuracy greater than 50%. We suspect that this low accuracy is a result of the interconnectedness of our feature vector. As Naive Bayes relies on an assumption of independence between variables within the feature vector, and nature is somewhat infamously non-independent [**?**], we remain largely unsurprised by this result.

It is perhaps unsurprising that, across all of the similarly accurate models, the same features kept appearing as the highest weights

in the weight vectors of our various predictors. Our neofeatures (difference between elevation and hydrology distances) were right up at the top, near Elevation, Roadway proximity, Hillshade, Fire Point proximity, and both Aspect values.

Combined, the ensemble of these models reached 85% accuracy on our validation set, but still sat near 75% on the Kaggle leaderboard. When we add in our second-level Random Forest classifier for cover types 1 and 2, our validation accuracy jumps to 91%, and our Kaggle score clocks in at approximately 80%.

5. CONCLUSION

In this paper, we examine a number of machine learning algorithms as they apply to the problem of classifying forest cover based on cartographic variables. We specify accuracy both for naive solutions (before tuning), and for solutions with more carefully tuned hyperparameters. The final collection of k-Nearest Neighbor, Support Vector Machine, Gradient Boosting Machine, and Random Forest result in a significant increase in accuracy over the next best approach, at a final accuracy of 80%.

In the course of our experimentation, we discovered a number of interesting relationships between the variables in the feature set. We were able to make use of some of these relationships, by introducing new features while transforming others, but some of the correlations between features remains a mystery. If we had the opportunity to continue work on this project, we would like to undertake a more comprehensive and formal analysis of the data.

For instance, what kind of relationship exists between the various Hillshade values? How closely correlated are they with Elevation? We have a great many soil types, but some of them seem to overlap. Do all soil types classified as "extremely stony" occur with similar environments, regardless of whether the soil is Bross family or Leighcan-Moran family? We would be interested in answering these questions and more, perhaps in consultation with an actual expert in the domain of geology.

6. ACKNOWLEDGMENTS