Predicting Forest Cover for Campground Planning

Project Members: Alex Simonoff, Rob Hammond, Anna Khazan
Project Group Advisor: Brian D'Alessandro
DS-GA 3001: Machine Learning

Introduction

This project's objective is to design a multiclass classification model for predicting forest cover types for plots of land in Colorado's Roosevelt National Forest. The plots of land are 30 meter by 30 meter cells, classified as one of seven possible cover types by the US Forest Service (USFS). Additional soil and cartographic variables were pulled from data collected by the US Geological Survey (USGS) and the USFS. We framed the problem as a task to classify regions based on characteristics of the cover types that make them more favorable for camping, and wanted to identify the forest types that are less likely to contain bears and are more likely to provide favorable conditions for campgrounds.

Krummholz, aspen, and Douglas fir forests provide habitats for black bears. Lodgepole pine forests are often home to black bears and even the occasional grizzly bear. Spruce-fir, cottonwood/willow, and ponderosa pine forests are home to smaller mammals and are unlikely to contain bears. The krummholz cover type contains abnormal vegetation that does not provide much protection from the elements making it very unfavorable for camping. Cottonwood/willow forests are often sub-irrigated regions that do not provide a suitable foundation for a camping tent. Spruce-fir forests are filled with flora and fauna (that do not include bears) that make them great for sightseeing and the vegetation is such that campers would be protected from the elements and supported by the soil foundation.

Data Source

We used data from Kaggle's Forest Cover Type Prediction Competition[1], which is broken up into a training set and a testing set. There are 15,120 observations in the training set, with an equal distribution over the seven forest cover types, consisting of 2,160 samples each. The testing set consists of 565,892 observations to classify, where model scores are computed via submissions on Kaggle as we did not have direct access the true classes for the test set.

For each training example there are 10 continuous, unscaled features for: elevation, aspect (degrees azimuth that a slope faces), horizontal distance to hydrology (surface water features such as rivers, lakes, etc.), vertical distance to hydrology, horizontal distance to roadways, horizontal distance to fire (wildfire ignition points), and hillshade at 9am, noon, and 3pm (0-255

[1] https://www.kaggle.com/c/forest-cover-type-prediction

index measured at the summer solstice), where all distances are in meters. There are four wilderness areas (the Rawah Wilderness Area, the Neota Wilderness Area, the Comanche Peak Wilderness Area, and the Cache la Poudre Wilderness Area) each described by a distinct, binary feature, as well as 34 soil-types represented in the same way. There were no missing data values to impute, as Kaggle provided a cleaned dataset to work with.

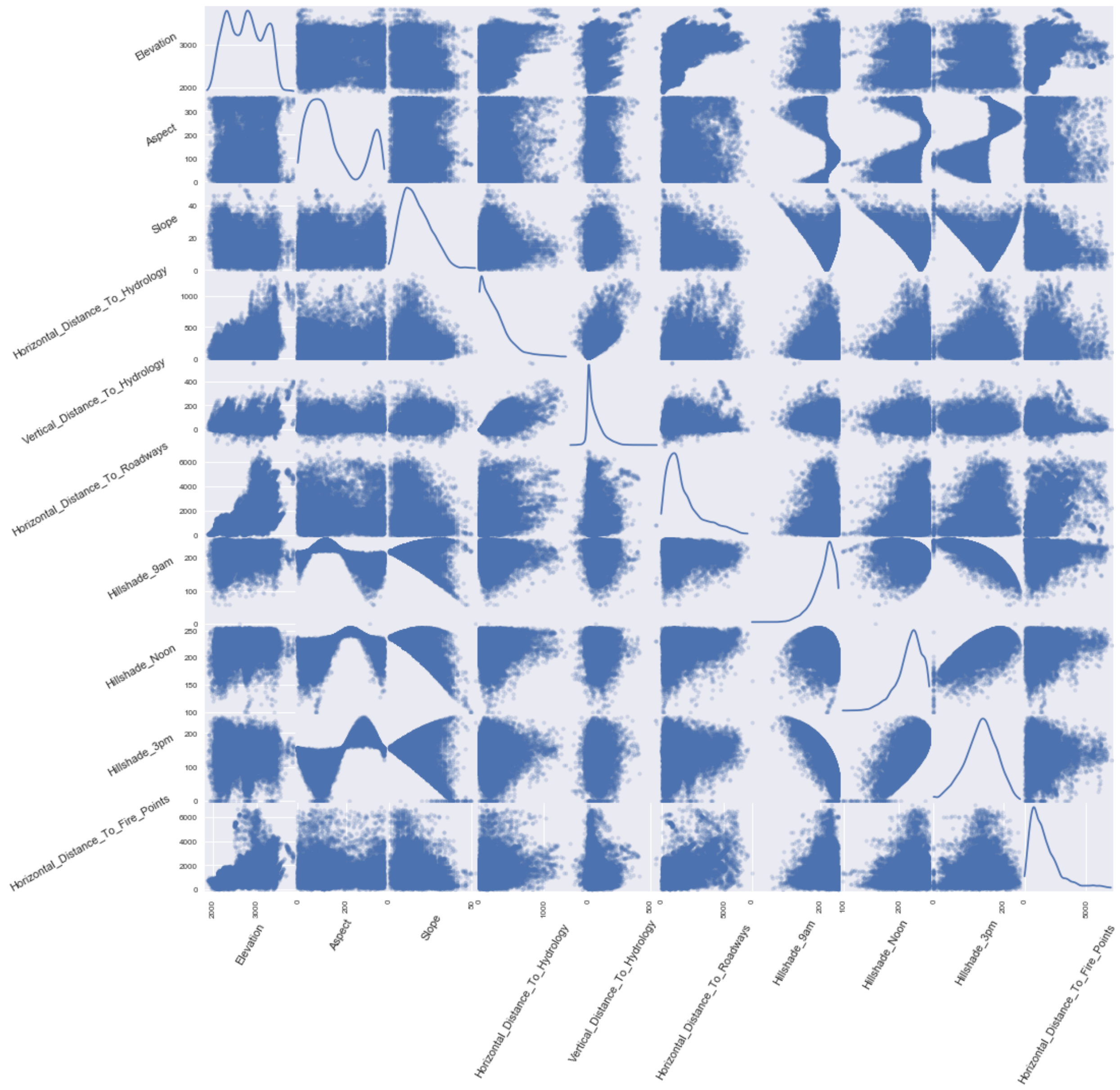


Figure 1: Bivariate distributions of each continuous feature in the original dataset with the kernel density estimator on the diagonal.

Feature Engineering

To start, we calculated several new features based on the given data to better describe the cover types. Much of our intuition was deduced from further research on the forest cover types and soil types. We additionally considered factors from the distributions in Figure 1, and where possible reduced our feature set to include only those that could provide useful predictive value.

Absolute degrees azimuth to True North

The initial feature related to each cell's aspect was provided on a range from 0 to 360 degrees. We decided to create a new feature transforming the aspect into absolute degrees from True North to reflect the fact that, for instance, 90 and 270 degrees in the original input may exhibit similar patterns due to symmetry, which means they may have direct sunlight for similar durations over the course of the day.

Distance above sea level

The original dataset provided information on each cell's vertical distance to hydrology, with a range that included some negative numbers that reflected cells that were located below sea level. We added a binary feature that indicated whether or not the forest cell was above or below sea level to create further distinction between cells that fell into these two categories. This feature was guided by the intuition that runoff from local hydrology may disproportionately affect forests in lower-lying regions.

Elevation buckets

In Figure 1 we noticed that the kernel density estimation plot of elevation was trimodal with distinct peaks between two valleys at 2,600 and 3,100 meters. Based on this, we created three binary features for each of the three elevation groups: 0 to 2,600 meters, 2,600 to 3,100 meters, and 3,100 to 8,000 meters. The goal was to treat introduce this as a categorical feature that treated samples with similar elevations as part of the same group.

Euclidean distance to hydrology

Given the vertical and horizontal distance to hydrology, we calculated the Euclidean distance between these two features to determine the direct distance to hydrology for each cell as a continuous feature. This new feature may have a more significant impact on the type of flora and fauna that can inhabit any particular forest, as the distance that animals must travel to get to the closest water source is defined by this direct distance more so than by the vertical or horizontal distance alone.

Soil description binary features

Kaggle provided detailed descriptions for each of the 40 soil types, with information such as the soil family, consistency, and stoniness. We translated these descriptions into 39 additional binary fields, and joined them to the original dataset. This allows us to group samples by groupings of soil families and types, rather than treating each soil type as a completely distinct feature.

First level interactions between every variable

We also calculated the first-level interaction between every pair of features that was not mutually exclusive. For instance, we multiplied together a cell's elevation and aspect, but ignored the interaction between wilderness types because only one of them could be nonzero for any given cell.

Removing Features

Creating all of these features increased our feature space to an approximately 15,000 x 5,000 dimension matrix. Any complex model with hyperparameter tuning would be intractable to run on a laptop in a reasonable amount of time, so we reduced the dimensionality by removing features with zero standard deviation as they do not capture any variance, and therefore provide no value to a model, which resulted in a feature space of approximately 2,800 features.

Feature Selection

In order to save on computation time (as 2,800 features run on a single support vector machine (SVM) takes several hours) we chose to perform two types of feature selection. Using the original data and MinMax scaling (0-1 scaling), standardized scaling, and normalization on our continuous features. We then ran our interacted and feature engineered data through random forests, extra trees, XGBoost, select percentile, and random feature elimination classifiers.

After fitting each model we stored the feature importances, sorted the features by the median feature importance they earned across the models and output the top 100. We ran into memory issues running feature selection on 2,800 features, and therefore we broke the data into four equally sized subsets. For each of the four subsets we performed the aforementioned process to determine the top 100 features. We then repeated the process one last time on the 400 features determined by combining the top 100 from each subset.

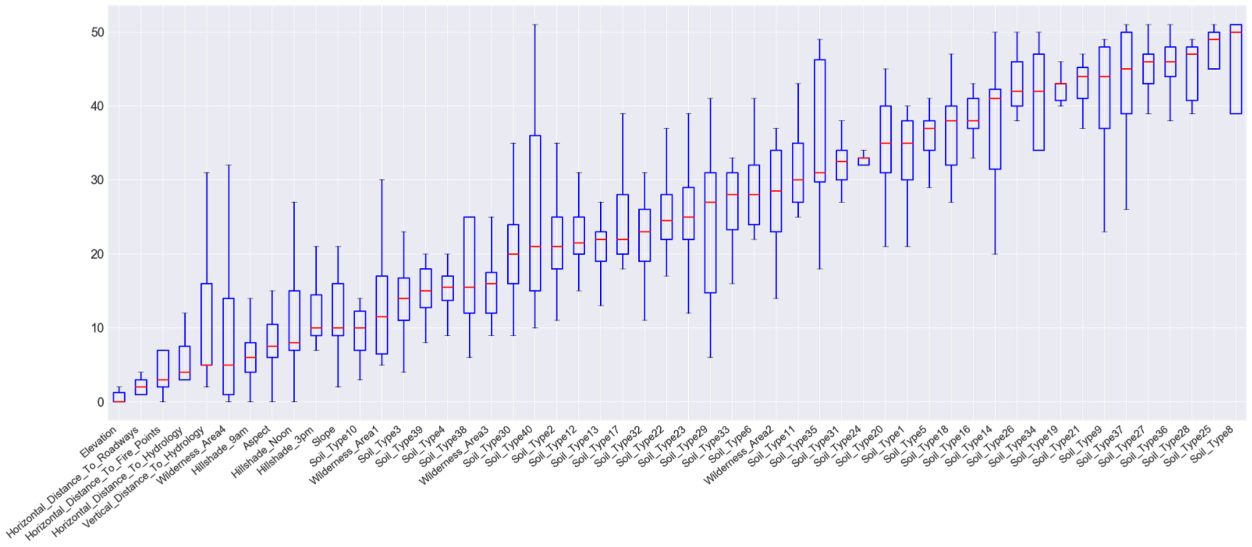


Figure 2: Boxplot of median feature importance of the original 54 features run over the 72 model combinations.

Model Implementation

We split the provided training dataset of 15,120 observations into training and validation datasets, and tested our models on a testing dataset of 565,892 observations using Kaggle's submission API. For each of our models we trained on the original dataset of 54 features, the engineered dataset of 96 features, and the dataset of the top 100 features when considering interactions. For certain iterations of our baseline algorithms, we used a 0-1 scaling on our continuous features. This particular form of data scaling ensures our models will be robust to features with low variance. This method also preserves sparsity across sparse features, of which we had several, especially among features describing the various soil characteristics. In the baseline algorithms that we tested we saw drastic improvements in model scores between scaled and unscaled datasets (see Table 1 in the Results section).

Baseline Models

We first tested versions of our data using logistic regression and SVMs to provide a benchmark for further model analysis. We separated the training dataset into two datasets to be used for model fitting and model validation, based on a 90% - 10% split. We then computed predictions for the full testing dataset to obtain model scores on Kaggle.

After our initial data exploration, we saw distinct patterns in relationships between certain features and feature combinations and the resulting forest cover type, leading us to believe that we could model linear relationships that logistic regression would take advantage of to produce a reasonable output. All iterations of this model ran very quickly, generally in under a few seconds, giving them a slight advantage over more advanced models. While a full grid search over many parameter choices proved to be prohibitively slow, we separately tried multiple parameter combinations. In the end we decided that the L2 penalty allowed for maximum results, as it was robust to outliers and did not lead to a sparse solution set of features, but distributed weights across more parameters.

We also ran multiple SVMs on the same group of datasets prompted by the fact that, while linear patterns were certainty evident, there may not have been a set of original or derived features that would allow logistic regression to decisively define a linear separating boundary. For our hyperparameter tuning, we decided to use an L2 penalty with a one-vs-rest approach to determining the best of multiple class options. This model was based on a squared hinge loss function.

To account for the fact that the L2 penalty would likely place nonzero weights on many of our features, we also ran both models on selected features based on their ranked importance. This ensured that even without the sparsity offered by L1 regularization, we still produced a model that was predicting based on relevant features.

Tree Based Models

After creating our baseline models we realized there was a lot of room for improvement in our Kaggle scores being in line with chance. This led us to choose three tree based models: random forests, extra trees, and extreme gradient boosting (XGBoost). Both random forests and extra trees classifiers use tree averaging, but random forests uses “intelligent” subsampling, and extra trees uses randomized subsamples to create tree ensembles. Because extra trees uses a randomized subsample it is much more efficient to train a model. However, there does not appear to be a way to know theoretically or empirically if random forests or extra trees will perform better without attempting them both. Similarities also exist between random forests and XGBoost in that their tree ensembling is the same, but XGBoost is an additive model as opposed to an averaging model, which should, in theory provide a competitive edge. Because each of the three models are either state of the art, highly efficient, or somewhere in the middle we decided that we would test all three models and compare our results.

We were able to make substantial improvements over our baseline model by using grid search over a number of hyperparameters for the XGBoost, random forests, and extra trees. Similar to our baseline, we had to carefully choose our data and the number of hyperparameters to ensure our models would be able to run on a laptop.

For our random forests model and XGBoost model we used grid search with both five-fold and ten-fold cross validations to test whether there was a discernible effect on the efficacy of our models that could validate the additional runtime. As we had run our models overnight we decided without prior knowledge of our results that we would use ten-fold cross validation for the extra trees classifier under the assumption that it would perform better. Empirically, using between five-fold and ten-fold shows a reduction in both the bias and variance of model estimates, so we decided either one would be a reasonable approach. In most instances when timing a single run using grid search we found that the runtime was around twice as long with twice as many folds in the cross validations, with little or no difference when comparing the accuracy of the best model on both the mean accuracy scores and the Kaggle scores. In addition, we found that the model variance tended to increase as we went from five-folds to ten-folds. After discovering the insignificant difference in model accuracy between XGBoost and random forests, we would choose five-folds in the future as there will be less model variance and lower runtime without significant losses in model accuracy allowing for a less costly model.

When using grid search for XGBoost we tested using a “max_depth” of 1, 3, and 5 nodes, a “min_child_weght” of 1, 3, and 5, a “learning_rate” of 0.15, 0.1, 0.075, and 0.05, 300 “n_estimators”, a “subsample” of 0.8, “colsample_bytree” of 0.8, and the multiclass softmax objective function with accuracy as our scoring function. The decision to test on these parameters was to model the default parameters, and values in the immediate vicinity as our reading through a number of examples had suggested empirically that these should give us the best model fits. Our best hyperparameters for grid search for each of the datasets was a “min_child_weight” of 1, a “max_depth” of 5, and “learning_rate” of 0.15; this lead to our top Kaggle score of 0.75907 on the top 100 features.

When running grid search on random forests, we tested using a “max_depth” of 1,3, and 5, “n_estimators” of 50 100, 200, 300, 500, 700, 800, and 1000, “criterion” of “gini” and “entropy”, and “max_features” of “sqr”’ and “log2” with the same objective and scoring function as XGBoost. Our decisions on choosing hyperparameters to test was again in the same fashion of reading through various kernels and examples and still optimizing the amount of time to train our models. Our best hyperparameters for the random forests produced a Kaggle score of 0.54041 with ten-fold cross validation using entropy for our split quality criterion, square root for our “max_features”, 1,000 “n_estimators”, and a “max_depth” of 5.

After seeing how long our models took to run on grid search, we were more pointed in our hyperparameter testing with our extra trees classifier, limiting the “min_samples_split” values of 2, 3, and 5, “max_features” of square root and log base 2, and “n_estimators” of 50, 100, 200, 300, 500, 700, and 1000 with 10 fold cross validation and an accuracy scoring function. Our best hyperparameter combination for extra trees with the top 100 features dataset produced our best score of all the models that we tested, and can be seen highlighted in yellow in Table 1, below. The optimal model used square root for the maximum features, a minimum of two samples for splitting, and 1,000 estimators.

Results

Model	**Dataset**	**Trasform**	**Kaggle Score**
Logistic Regression	Original	None	0.17989
Logistic Regression	Original	0-1 Scaling	0.54544

Logistic Regression	Engineered	None	0.47318
Logistic Regression	Engineered	0-1 Scaling	0.54073
Logistic Regression	Top 100 Engineered with Interactions	None	0.45303
Logistic Regression	Top 100 Engineered with Interactions	0-1 Scaling	0.54668
SVM	Original	None	0.02687
SVM	Original	0-1 Scaling	0.53176
SVM	Engineered	None	0.05936
SVM	Engineered	0-1 Scaling	0.52973
SVM	Top 100 Engineered with Interactions	None	0.04755
SVM	Top 100 Engineered with Interactions	0-1 Scaling	0.55526
Random Forests	Original	None	0.51607
Random Forests	Engineered	None	0.53788
Random Forests	Top 100 Engineered with Interactions	None	0.54041
XGBoost	Original	None	0.73838
XGBoost	Engineered	None	0.74069
XGBoost	Top 100 Engineered with Interactions	None	0.75907
Extra Trees	Original	None	0.75748
Extra Trees	Engineered	None	0.73773
Extra Trees	Top 100 Engineered with Interactions	None	0.79302

Table 1: Summary of model results across models, features, and data transformations with our best performing model highlighted in yellow.[2]

Discussion

Although lower than the tree based models, we still saw drastically improved baseline model performance on scaled data that on any subsets of features in their original input form. The SVM was most affected by unscaled data, producing scores of less than 0.06, regardless of the

[2] Please note: our best performing model, highlighted in yellow, placed in spot 219 of the publicly submitted scores, which is in the top 13% of all scores including those having used cheat methods. Also note that the competition is over, so this ranking is a comparison of our score to the scores submitted during the submission period.

feature subset that was selected. Logistic regression was much less influenced by the unscaled data, and showed significant improvement between the original, unscaled data and the engineered, unscaled data; the model score went up from .17989 to .47318, with a slight decrease in performance when the dataset was limited to only the top 100 most important features.

Baseline model scores were relatively consistent on scaled data, regardless of model and input dataset. This seems to show that feature ranges have a significant impact on these types of models, and the important features will have larger weights placed on them in the resulting model. Whereas the model scores on the scaled data were very close, running models on only the top 100 features did produce the best results, with scores of .54668 with logistic regression and .55526 with SVM.

A few observations from our XGBoost testing that should be noted are that our Kaggle score for five- and ten-fold cross validations produced the exact same score with near identical mean accuracy scores in grid search for the best parameters. In addition to there being no substantive gain in accuracy between five- and ten-fold cross validation, we found there was less than 0.5% gain in our scores when comparing the original features and the base features, and only a 2% gain in our score when using our top 100 feature dataset. We can also see in Figure 3, below, that XGBoost should be run for more epochs, but there was no effective way to control for this aside from creating our own implementation of the algorithm. This early stopping from the model is also likely the reason why a high learning rate was able to perform so well.

Much like our XGBoost model there was very little gain in using our enhanced feature datasets when comparing our resulting Kaggle scores. Similarly there was no significant gain in using ten-fold cross validation in comparison to five-fold cross validation. However, for our extra trees model we had a substantial improvement (4%) when using our top 100 features dataset compared to the dataset provided by Kaggle. An item of concern that we encountered was that random forests seemed to perform particularly well with this dataset for other people, so there would need to be further consideration for choosing hyperparameters and controlling the number training epochs for a future implementation.

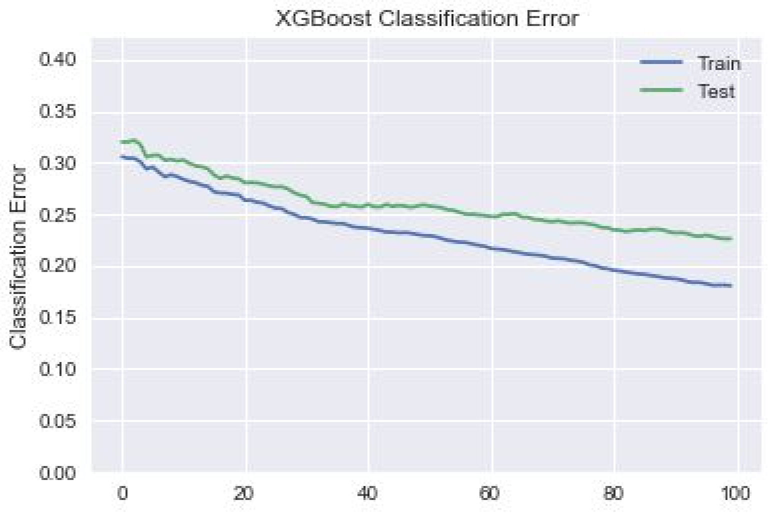


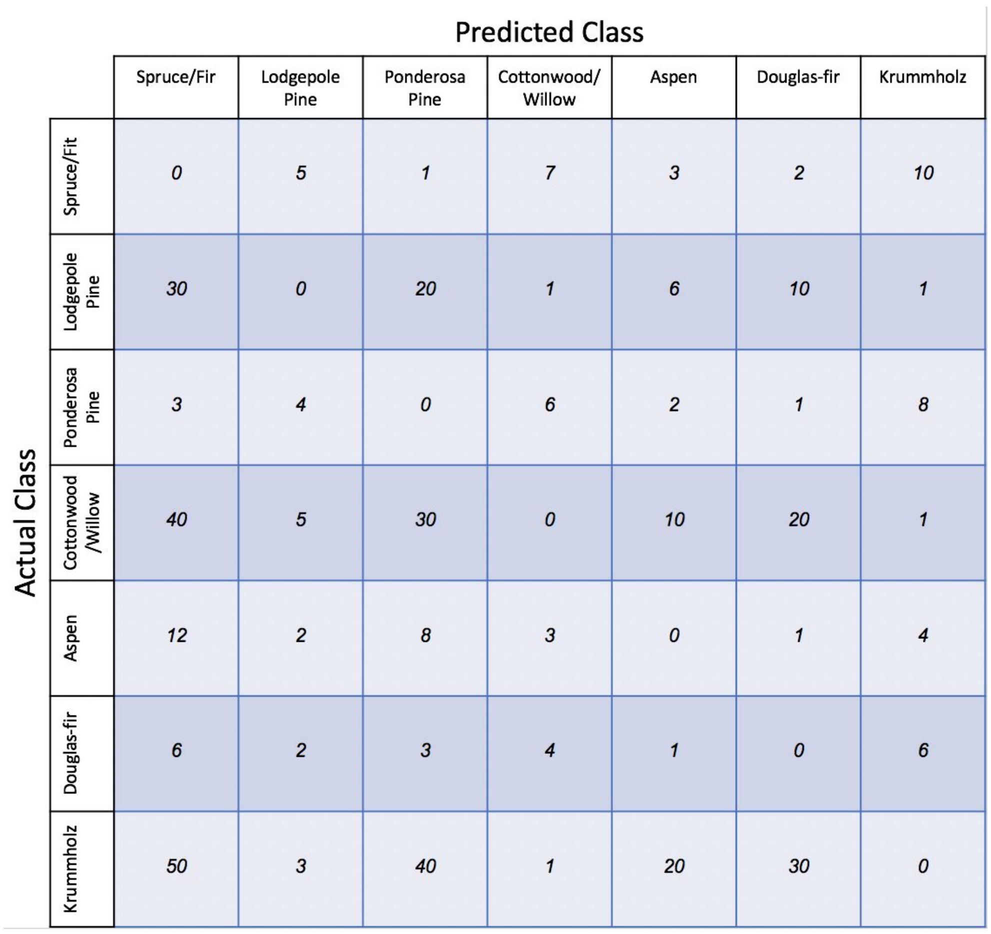
Figure 3: XGBoost training for 100 epochs with a multiclass softmax objective function, multiclass classification error, and default hyperparameters.

Next Steps

Weighted Misclassification Errors

For our objective of predicting optimal locations to site campgrounds, misclassification errors are not consistent across all instances. A useful next step would be to implement a loss function that optimizes algorithms based on pairs of predicted and actual classes. This can be accomplished by producing a confusion matrix, an example of which is detailed in Table 2 below, which assigns a specific weight to each type of misclassification error. The winning model would then be the one that produces the lowest scores based on this derived metric. In our example of trying to assess forest cover type based primarily on the availability of suitable vegetation and tree cover and lack of black bears, misclassification errors are not evenly distributed throughout the confusion matrix.

For example, predicting a spruce/fir or ponderosa pine cover type, known for very few bear sightings and ample protective vegetation while the true class is krummholz, a forest cover type hospitable to black bears with little protective vegetation, would be a very costly mistake. The corresponding cells in the confusion matrix therefore contain some of the highest weights. If the opposite misclassification were to occur, this would lead to a much lower misclassification weight, since the surprise presence of bears is more problematic than the surprise lack of suitable vegetation for a campground. Of note are the diagonal cells in the below matrix, which ensure that every correctly classified sample gets a weight of 0 added to the summed total.



Actual Class	Predicted Class: Spruce/Fir	Lodgepole Pine	Ponderosa Pine	Cottonwood/ Willow	Aspen	Douglas-fir	Krummholz
Spruce/Fit	*0*	*5*	*1*	*7*	*3*	*2*	*10*
Lodgepole Pine	*30*	*0*	*20*	*1*	*6*	*10*	*1*
Ponderosa Pine	*3*	*4*	*0*	*6*	*2*	*1*	*8*
Cottonwood /Willow	*40*	*5*	*30*	*0*	*10*	*20*	*1*
Aspen	*12*	*2*	*8*	*3*	*0*	*1*	*4*
Douglas-fir	*6*	*2*	*3*	*4*	*1*	*0*	*6*
Krummholz	*50*	*3*	*40*	*1*	*20*	*30*	*0*

Table 2: Confusion matrix showing weights corresponding to misclassifications across classes.

Principle Component Analysis (PCA)

Rather than ranking all the available features and then choosing the top few as inputs to a model due to time and memory constraints, another useful next step will be to run PCA on the feature set to determine the underlying structure that can capture the maximum amount of variance in the data. This would allow us to reduce the number of dimensions a model is fitting to just the supporting vectors, simplifying the overall analysis. This approach would also alleviate some of the time and storage constraints we experienced when attempting to run our models on datasets that had too many features.

Conclusions

With each of our models there was varying improvement when using our dataset with feature engineering as opposed to the base dataset provided by Kaggle. The improvements in our baseline models are due to the robust feature set used in the models, which allowed for more ways to better understand the data. This is not surprising because we explicitly captured certain feature interactions and created a more explanatory feature set. This, however, comes with the caveat that with our nonlinear, tree based models, we did not have a significant difference in overall performance. If implementing such a model in production this would not justify the amount of time spent engineering features, transforming data, and increasing runtime for twice as many features as the original dataset. The reason that our nonlinear models did not perform substantially better is likely due to the fact that they perform their own version of feature engineering when building-out the trees and ensembling them, which leaves little room for improvement with our feature engineering.

Contrary to our expectation that XGBoost would be the best performing model, the extra trees classifier turned out to be our best performing model. Our expectations were that a state of the art algorithm like XGBoost would offer the best performance, but from our experience it seems that the implementation of the model did not allow for enough training epochs to be able to reach an asymptote in its learning. This was evident in its high learning rate, because it is likely that a model that learned more slowly over a longer period would be able to outperform the quick learning model. The same reasoning could also apply to the performance of our random forests models as they did not reach our expected performance rate. Going forward we would narrow our modeling to use five-fold cross validations, implement our own XGBoost, train for longer with random forests, and only consider the base dataset. With the reduced number of modeling parameters to control for we would have more computational capacity to be able to do a more pointed search on our hyperparameters; this would be able to improve all three of our nonlinear models in order to obtain a better end product that aligns with our expectations of how each model should perform.