What's covering me? - Prediction of forest cover type based on cartographic variables

Devin Austin **Duncan T. Tulimieri**

1 Introduction

National parks are some of the most beautiful places on this little blue planet we call home. They are some of the only remaining "untouched" land that allows nature to flourish. These parks, while untouched, are not ignored. In fact, they are some of the most studied areas around. Parks can range from a few thousand to over 52 million acres [2].

The goal of this project was to apply various machine learning techniques to a large data set that would reflect common data sets seen by data scientists in the industry. We chose the Forest cover type data set from the University of California Irvine's machine learning repository because it contains a large amount of both continuous and categorical predictors and it was recently used in a Kaggle Playground Prediction Competition, where teams compete to build the best predictive model.

This data set utilized information from four parks (Rawah, Comanche Peak, Neota, and Cache la Poudre) in the Roosevelt National Forest in north Colorado. The geographical layout of these parks can be seen in figure 1. Only cartographic predictors taken from geological surveys of the parks were used in this data set. The predictors include elevation, soil type, slope,hill shade at various times of day, and distance to the nearest surface water feature. The sought classification of these features was the forest cover type, which refers to the predominant tree species in a given region. All forest cover types along with a full list of predictors can be seen in figure 2. The data set contains 581012 observations, 54 predictors, and 7 classifications of forest cover type.

Utilizing several machine learning algorithms and ensemble machine learning techniques discussed in class, we sought to build an algorithm that would achieve high predictive power when classifying a portion of the overall forest cover type data set.

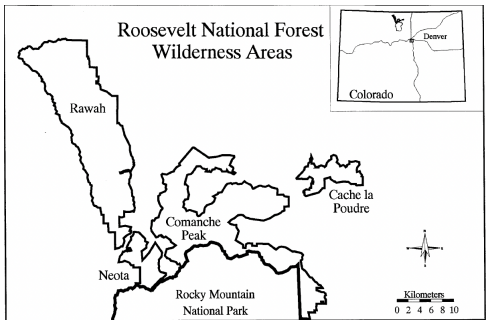

Figure 1: Study area location map. Taken from Blackard and Dean (1999) [1].

1.1 Previous literature using Forest Cover Type Data set

There have been several studies that use various machine and deep learning algorithms and methods to classify the forest type [1, 4, 6]. These articles used different algorithms and methods ranging from

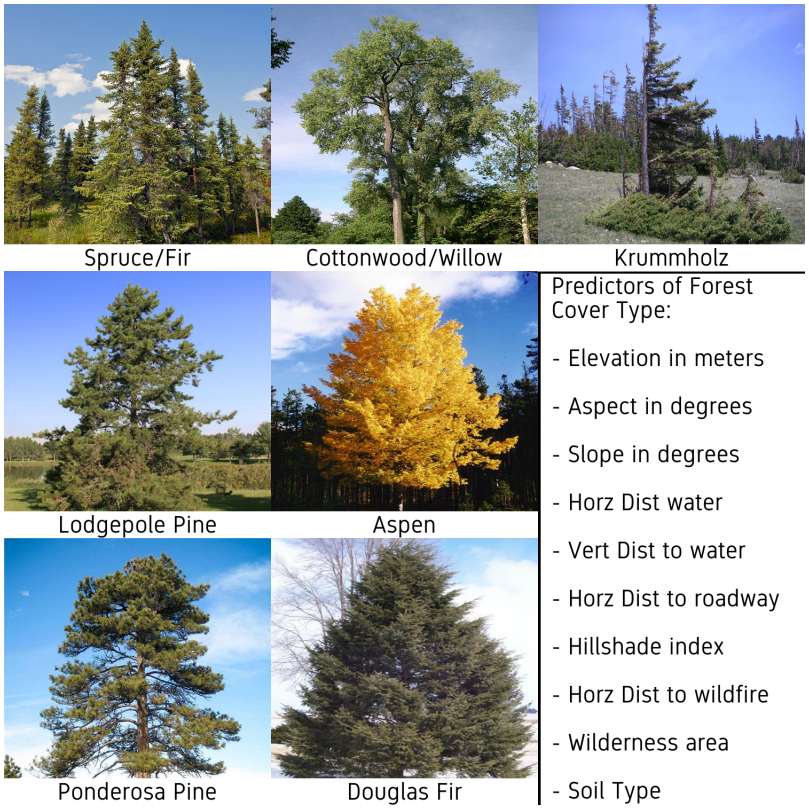

Figure 2: Forest Cover Types and Predictors

Number of independent variables	Description of variables
54	Ten quantitative variables + four wilderness areas + 40 soil types
53	Same as '54' but excluding distance-to-wildfire-ignition-points
21	Ten quantitative variables + 11 generalized soil types
20	Same as '21' but excluding distance-to-wildfire-ignition-points
10	Ten quantitative variables only
9	Same as '10' but excluding distance-to-wildfire-ignition-points

Table 1: Number of input variable subsets examined. Taken from Blackard and Dean (1999) [1].

neural networks to bagging and boosting. The accuracy of these algorithms and methods ranged from ~50% to ~74%.

1.1.1 Black and Dean (1999) [1]

Blackard and Dean were the first to publish on this data set [1]. These authors compared the performance of a neural network, linear discriminant analysis, and quadratic discriminant analysis on various subsets of the entire data set. The authors split the data set into 6 subsets of the entire data set (Table 1). The authors split the data set because they had *a priori* ideas about which predictors would hold large predictive power.

In order to train the best neural network, the authors did many iterations of editing model parameters. The neural network was initialized and kept to one input layer, one hidden layer, and one output layer. The layers were dense and no dropout was used. The authors systematically changed the number of nodes in the hidden layer. This was done to determine the best learning and momentum rates. This network was trained using back propagation to update the weights. The weights for this model were initialized by randomly sampling a uniform distribution between negative one and positive one. The hidden layer used linear activation functions, while the output layer used logistic activation functions. After the authors manually found parameter specifications they believed yielded the best model, to

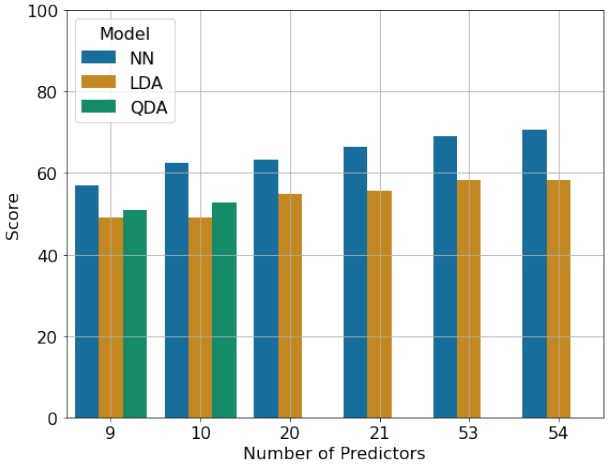

Figure 3: Comparison of artificial neural network and discriminant analysis classification results. Republished from Blackard and Dean (1999) [1].

ensure these were the best parameters, they produced thirty new neural networks with the previously found parameters and set initial random synaptic weights. This process was used to ensure the weight space was fully explored due to the stochastic nature of initializing weights. A neural network with 54 input nodes, 120 nodes in the hidden layer, 7 output nodes, a learning rate of 0.05, and a momentum rate of 0.5 wad determined to be 'optimal'. This model led to the a classification accuracy of 70.58%.

The authors utilized both linear and quadratic discriminant analyses. The quadratic discriminant analysis model was unstable as qualitative and discrete variables were added to the predictor matrix. Of all of the subsets of data tested, the linear discriminant analysis model had the highest classification accuracy (58.38%) with all (54) predictors. The quadratic discriminant analysis model achieved a classification accuracy of 49.15% with 10 predictors, but then became unstable.

Overall, the authors were able to achieve high predictive power. All the models tested were able to perform better than chance (Figure 3). The neural network achieved the highest overall accuracy (~71%) when the model included all the variables. The neural network showed steady increase in accuracy as more predictors were added. The discriminant analyses also showed increases in accuracy as more predictors were added.

The authors pose many reasons for the discriminant analyses models performing worse than the neural network. One potential reason for this discrepancy could be due to the underlying assumptions of the discriminant analyses models. Discriminant analysis models assume multivariate Gaussian distribution and can also assume shared covariance matrices between all classes (linear discriminant analysis only). A neural network has no underlying assumptions about underlying distributions and therefore will be more flexible in modeling data. The authors also state that another reason discriminant analyses could have performed worse than neural works would be due to the non-linearity of the data. The discriminant analyses typically perform well with linear data while the neural network is again flexible in regards to linearity of data. One area where the discriminant analyses outperformed the neural network is in computational time. The discriminant analyses took 5 hours to run, while the neural network, once finalized, took 45 hours to finalize. The 45 hour run time of the neural network also does not take into account the time needed for the operator to tune all the hyperparameters.

1.1.2 Oza and Russell (2001) [6]

In 2001, Oza and Russell[6] utilized the forest cover data set as one of many large data sets set to validate alternative versions of traditional ensemble learning algorithms that use multiple machine learning techniques to increase overall predictive performance. The authors first created an online version of bagging, which traditionally used the predictive power of several models given randomly re-sampled subsets of data; however, the online version enables weights on any given sample drawn from a binomial Poisson distribution. Additionally, the authors created an online version of boosting,

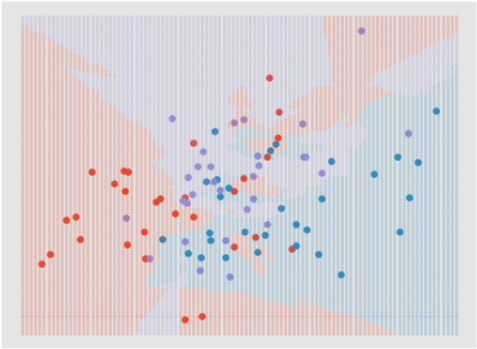

Figure 4: KNN classifier on simulated data. Each class is represented by a single color. Shaded regions show predicted value for new potential instance.

which traditionally uses training data that is first randomly re-sampled with replacement, then used to train a model. All training samples that are misclassified by the first model are given weights, which increase the probability of misclassified points being sampled for subsequent models. While traditional boosting determines re-sampling weights based on performance on the entire training data set, online boosting determines only uses samples seen before. The authors report that their methods were just as effective while being quicker than traditional bagging and boosting. The authors prove that ensemble learning techniques are effective when applied to the forest cover data set; however, they do not report model scores for the forest cover data set.

2 Algorithms

2.1 K nearest neighbors (KNN)

K-nearest neighbors (KNN) is a powerful method for both classification and regression data sets. It operates on the assumptions that the data exists in a feature space that allows the measurement of some degree of similarity between samples. This similarity is often described as the distance between two points and is reliant on a single *k* value. The k value determines how many of the closest points are considered when classifying a given sample. For example, if k is equal to three than the KNN algorithm will considered the three closest training points to the sample and select the classification based on a majority vote. Another parameter commonly given to the KNN algorithm is to enable weights on individual points. The weights are determined by the distance between points, so closer points have a stronger vote during classification. A brief outline of the KNN algorithm can be seen in algorithm 1 and a simulated two dimensional example of the KNN algorithm with k=5 and ...

Algorithm 1: K-Nearest Neighbors

Input: X_{train}, y_{train}, X_{test}, k, $weights$
Output: Prediction
begin
  for *i, x in enumerate(*X_{test}*)* **do**
    distances = evaluate distance between x and all X_{train} samples
    nearest neighbor classifications = choose k closest samples of X_{train} to x, and find their respective classifications in y_train
    Prediction[i] = mode[nearest neighbors classifications] evaluated using weight parameter

2.2 Support Vector Machine (SVM)

A support vector machine (SVM) is a useful machine learning algorithm for classifying data sets.The algorithm attempts to find a hyperplane that can distinctively split several classes of data utilizing existing samples for a given class. These samples are known as support vectors and can consist

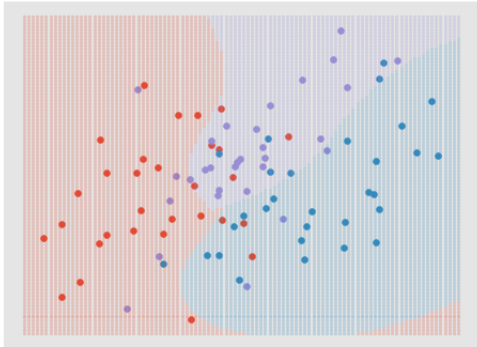

Figure 5: SVM classifier on simulated data. Each class is represented by a single color. Shaded regions show predicted value for new potential instance.

of samples both at the edge and within a given cluster of training data. One of the strengths of SVMs is that it can account for outliers in a training data set that may lead to misclassification. To accomplish this, the algorithm uses cross validation between all support vectors to find how many misclassifications need to be allowed in order to create the best model. One of the key tools SVMs uses to increase predictive power is with the help of kernels, which produce transformations on the data set in order efficiently separate classes. One weakness of SVMs is if multiple classifications overlap in the training set, SVMs will have trouble predicting the correct classifier. Another weakness of the SVM, which stems from the cross validation of all support vectors, is that it is inefficient for large data sets and produces very high run times. A brief outline of a binary SVM algorithm can be seen in algorithm 2 and a simulated...

Algorithm 2: Support Vector Machine

Input: X_{train}, y_{train}, X_{test}, C, $Kernal$
Output: Prediction
Let $Classes$ be the unique classes in y_{train}
begin
  for *i,x in enumerate(*X_test*)* **do**
    Hyperplane = create a vector of points that make up the hyperplane using a transformation based on $Kernal$ and with a penalty term C
    if *x exists on one side of hyperplane* **then**
      Predictions[i] = 1
    else if *x exists on other side of hyperplane* **then**
      Prediction[i] = 0

2.3 Linear Discriminant Analysis (LDA)

Linear discriminant analysis (LDA) is a powerful method for classifying data. That is, when the assumptions of the model hold true. LDA assumes that all the predictors are normally distributed. Naturally, this algorithm models the data as (multivariate) Gaussian (Algorithm 3). More specifically, each class has a mean (μ) for each predictor. However, each class does not have their own covariance (Σ) matrix. Each class is assumed to share a Σ matrix. These assumptions can help when there is enough data to get a reasonable estimate of central tendency, but not enough to get a stable estimate of dispersion within predictors and correlation between predictors. This algorithm performs well when these assumptions are not violated (Figure 6).

Like all models, LDA will perform poorly if the assumptions are not met. For example, if most predictors are normally distributed, but a powerful predictor has a Poisson distribution, its predictive power will go to waste. However, there are methods, such as standard scaling, that can fix this issue. Although, there can be another problem that isn't fixed so easily. For example, if all predictors within a class appear normal, but each class exhibits different relationship between and within the predictors,

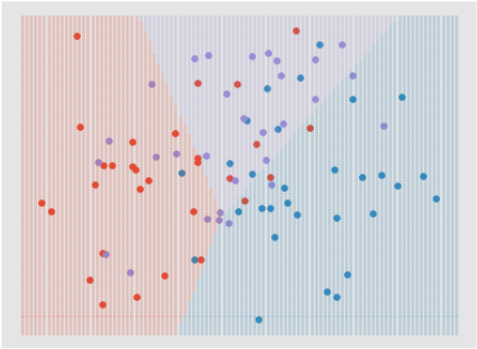

Figure 6: LDA classifier on simulated data. Each class is represented by a single color. Each classes simulated data comes from a unique μ vector, but shared Σ matrix, therefore not violating the LDA assumptions. Shaded regions show predicted value for new potential instance. Inspiration from Hastie et al. - *Elements of Statistical Learning* [5].

LDA will suffer again. And unlike violating the assumption of normality, violating the assumption that all classes share a covariance matrices, there are not any methods to fix this.

Algorithm 3: Linear Discriminant Analysis

Input: X_{train}, y_{train}, X_{test}
Output: Prediction
Let $Classes$ be the unique numeric classes in y_{train}
begin
  for *class in Classes* **do**
    MU[class,:] = average of all predictors for instances with class $class$ in X_{train}
  COV = covariance matrix of X_{train}
  for *i, x in enumerate(X_{test})* **do**
    for *class in Classes* **do**
      probability[class] = evaluate a multivariate normal distribution at x with parameters (μ = MU[class,:], Σ = COV)
    Prediction[i] = argmax[probability]

2.4 Quadratic Discriminant Analysis (QDA)

Quadratic discriminant analysis (QDA) is another algorithm for classifying data, but is more flexible in how it models the data compared to LDA. Both LDA and QDA model central tendency the same, with a μ vector for each class, but they model within and between dispersion Σ differently. Whereas LDA models Σ for all class together, QDA will model Σ for each class separately (Algorithm 4). This small difference will afford this algorithm much more flexibility around classes having different relationships between and within predictors (Figure 7). While this allows more flexibility, it comes at a cost. The cost is that there must be enough data in each class to allow reasonable estimation of within and between dispersion.

2.5 Logistic Regression

Logistic regression is a simple, yet effective, machine learning model (Figure 8). Whereas linear regression predicts continuous values along a line, logistic regression predicts continuous values along a sigmoid curve (such as a logistic function) (Algorithm 5). Prediction along a sigmoid curve allows for much better classification than along a line due to the sigmoid function having more values at 0 and 1.

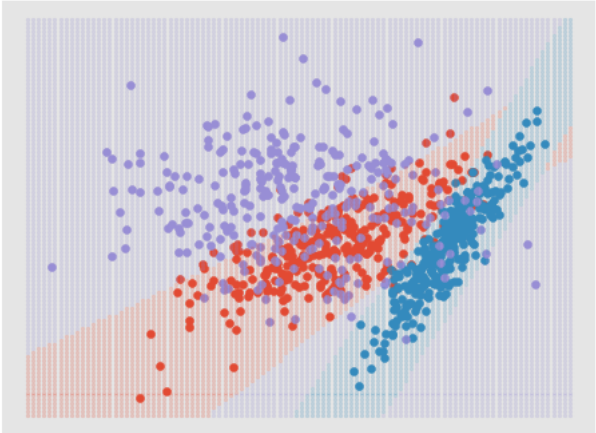

Figure 7: QDA classifier on simulated data. Each class is represented by a single color. Each classes simulated data comes from a multivariate Gaussian with a unique μ vector and a unique Σ matrix. Shaded regions show predicted value for new potential instance. Inspiration from Hastie et al. - *Elements of Statistical Learning* [5].

Algorithm 4: Quadratic Discriminant Analysis

Input: X_{train}, y_{train}, X_{test}
Output: Prediction
Let $Classes$ be the unique numeric classes in y_{train}
begin
  for *class in Classes* **do**
    MU[class,:] = average of all predictors for instances with class $class$ in X_{train}
    COV[class] = covariance matrix for all predictors of X_{train} with instances class $class$
  for *i, x in enumerate(X_{test})* **do**
    for *class in Classes* **do**
      probability[class] = evaluate a multivariate normal distribution at x with parameters (μ = MU[class,:], Σ = COV[class])
    Prediction[i] = argmax[probability]

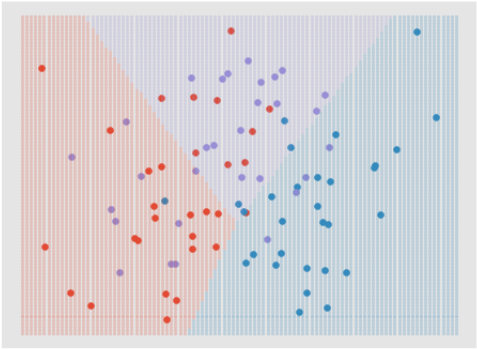

Figure 8: Logistic Regression classifier example. Each class is represented by a single color. Shaded regions show predicted value for new potential instance.

Algorithm 5: Logistic Regression

Input: X_{train}, y_{train}, X_{test}
Output: Prediction
Use an optimizer, like Newton's method, to find optimal coefficients (β) for logistic model based on X_{train}.
for *i, x in enumerate(X_{test})* **do**
  $probability$ = evaluate x using a logistic function with coefficients β.
  if *probability >= 0.5* **then**
    Predictions[i] = 1
  else if *probability < 0.5* **then**
    Prediction[i] = 0

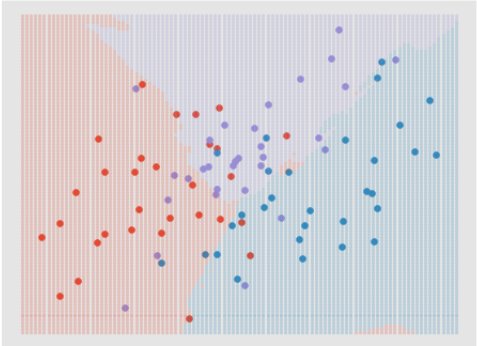

Figure 9: Ensemble classifier on simulated data. Each class is represented by a single color. The ensemble classifier is made from the previous models trained on the simulated data (e.g., KNN [Figure 4], SVM [Figure 5], LDA [Figure 6], QDA [Figure 7], and Logistic Regression [Figure 8]) and a majority vote for prediction. Shaded regions show predicted value for new potential instance.

2.6 Ensemble Learning

Ensemble learning is an machine learning technique that utilizes several models to produce an overall predictor that theoretically should have a better performance than any single model. These models can be the same learning algorithm with various versions of the training data set (e.g. bagging), they can be the same learning algorithm repeated with re-weighted samples of the training set based on previous model performance (e.g. boosting), or they can be several learning algorithms that are trained with the same data and whose predictions can be combined via an average for regression and majority vote for classification problems. For the latter example, utilizing several algorithms where each have their own various assumptions and biases produces an event where the algorithms account for each other's weaknesses.

Algorithm 6: Ensemble Learning

Input: X_{train}, y_{train}, X_{test}
Output: Prediction
begin
  M = base models (e.g. KNN, SVM, QDA)
  for *i, x in enumerate(X_{test})* **do**
    for *m in M* **do**
      ModelPredictions[m] = record prediction of x for model m
    Prediction[i] = find mode of $ModelPredictions$ to determine final prediction of x

		Predicted Class						
		SF_p	LP_p	PP_p	CW_p	A_p	Df_p	K_p
Actual Class	SF_A	$\frac{\Sigma 1(SF_A=SF_p)}{\Sigma 1(SF_A)}$	$\frac{\Sigma 1(SF_A=LP_p)}{\Sigma 1(SF_A)}$	$\frac{\Sigma 1(SF_A=PP_p)}{\Sigma 1(SF_A)}$	$\frac{\Sigma 1(SF_A=CW_p)}{\Sigma 1(SF_A)}$	$\frac{\Sigma 1(SF_A=A_p)}{\Sigma 1(SF_A)}$	$\frac{\Sigma 1(SF_A=Df_p)}{\Sigma 1(SF_A)}$	$\frac{\Sigma 1(SF_A=K_p)}{\Sigma 1(SF_A)}$
	LP_A	$\frac{\Sigma 1(LP_A=SF_p)}{\Sigma 1(LP_A)}$	$\frac{\Sigma 1(LP_A=LP_p)}{\Sigma 1(LP_A)}$	$\frac{\Sigma 1(LP_A=PP_p)}{\Sigma 1(LP_A)}$	$\frac{\Sigma 1(LP_A=CW_p)}{\Sigma 1(LP_A)}$	$\frac{\Sigma 1(LP_A=A_p)}{\Sigma 1(LP_A)}$	$\frac{\Sigma 1(LP_A=Df_p)}{\Sigma 1(LP_A)}$	$\frac{\Sigma 1(LP_A=K_p)}{\Sigma 1(LP_A)}$
	PP_A	$\frac{\Sigma 1(PP_A=SF_p)}{\Sigma 1(PP_A)}$	$\frac{\Sigma 1(PP_A=LP_p)}{\Sigma 1(PP_A)}$	$\frac{\Sigma 1(PP_A=PP_p)}{\Sigma 1(PP_A)}$	$\frac{\Sigma 1(PP_A=CW_p)}{\Sigma 1(PP_A)}$	$\frac{\Sigma 1(PP_A=A_p)}{\Sigma 1(PP_A)}$	$\frac{\Sigma 1(PP_A=Df_p)}{\Sigma 1(PP_A)}$	$\frac{\Sigma 1(PP_A=K_p)}{\Sigma 1(PP_A)}$
	CW_A	$\frac{\Sigma 1(CW_A=SF_p)}{\Sigma 1(CW_A)}$	$\frac{\Sigma 1(CW_A=LP_p)}{\Sigma 1(CW_A)}$	$\frac{\Sigma 1(CW_A=PP_p)}{\Sigma 1(CW_A)}$	$\frac{\Sigma 1(CW_A=CW_p)}{\Sigma 1(CW_A)}$	$\frac{\Sigma 1(CW_A=A_p)}{\Sigma 1(CW_A)}$	$\frac{\Sigma 1(CW_A=Df_p)}{\Sigma 1(CW_A)}$	$\frac{\Sigma 1(CW_A=K_p)}{\Sigma 1(CW_A)}$
	A_A	$\frac{\Sigma 1(A_A=SF_p)}{\Sigma 1(A_A)}$	$\frac{\Sigma 1(A_A=LP_p)}{\Sigma 1(A_A)}$	$\frac{\Sigma 1(A_A=PP_p)}{\Sigma 1(A_A)}$	$\frac{\Sigma 1(A_A=CW_p)}{\Sigma 1(A_A)}$	$\frac{\Sigma 1(A_A=A_p)}{\Sigma 1(A_A)}$	$\frac{\Sigma 1(A_A=Df_p)}{\Sigma 1(A_A)}$	$\frac{\Sigma 1(A_A=K_p)}{\Sigma 1(A_A)}$
	Df_A	$\frac{\Sigma 1(Df_A=SF_p)}{\Sigma 1(Df_A)}$	$\frac{\Sigma 1(Df_A=LP_p)}{\Sigma 1(Df_A)}$	$\frac{\Sigma 1(Df_A=PP_p)}{\Sigma 1(Df_A)}$	$\frac{\Sigma 1(Df_A=CW_p)}{\Sigma 1(Df_A)}$	$\frac{\Sigma 1(Df_A=A_p)}{\Sigma 1(Df_A)}$	$\frac{\Sigma 1(Df_A=Df_p)}{\Sigma 1(Df_A)}$	$\frac{\Sigma 1(Df_A=K_p)}{\Sigma 1(Df_A)}$
	K_A	$\frac{\Sigma 1(K_A=SF_p)}{\Sigma 1(K_A)}$	$\frac{\Sigma 1(K_A=LP_p)}{\Sigma 1(K_A)}$	$\frac{\Sigma 1(K_A=PP_p)}{\Sigma 1(K_A)}$	$\frac{\Sigma 1(K_A=CW_p)}{\Sigma 1(K_A)}$	$\frac{\Sigma 1(K_A=A_p)}{\Sigma 1(K_A)}$	$\frac{\Sigma 1(K_A=Df_p)}{\Sigma 1(K_A)}$	$\frac{\Sigma 1(K_A=K_p)}{\Sigma 1(K_A)}$

Table 2: Confusion matrix. Along the diagonal there are the correct classification and elsewhere are the incorrect classifications. Abbreviations: SF: Spruce/Fir, LP: Lodgepole Pine, PP: Ponderosa Pine, CW: Cottonwood/Willow, A: Aspen, Df: Douglas-fir, K: Krummholz, X_A: Actual X class, X_P: Predicted X class.

figures/all_models_scores_wbg.png

Figure 10: Model scores. Scores were calculated according to Equation 1.

3 Results

We set out to create a model that was able to very accurately predict forest cover type from 54 cartographic variables describing the area. We used multiple linear and non-linear classifiers from sklearn [7]. Since our goal was to obtain the highest score possible, we also combined (ensemble) all of the individuals models to create a model that predicted based on a majority vote of multiple models. Combining multiple models has been shown to improve performance [8]. The theory behind this idea is that each weak learner is learning something specific about the data, so when combined, each model can contribute to create a more stable and accurate prediction [8].

In order to assess how well our models performed we needed performance metrics. Our first metric of performance was accuracy (as seen in Equation 1 [7]). Accuracy gave us an overall idea of the model's performance. Although, in multi-class classification this can be a harsh way to measure true model accuracy [7]. Therefore, we sought out a way to understand our models on a deeper level to understand if there were classes that our models were either predicting very well or not predicting well. Therefore, we chose the confusion matrix as another performance metric (Table 2). Confusion matrices allow one to see the accuracy of a model at predicting each class (as seen in Equation 2). A completely perfect multi-class classifier would have a confusion matrix with 100% accuracy on the diagonal and 0% accuracy everywhere else. On the other hand, a completely inaccurate multi-class classifier would have a confusion matrix with 0% on the diagonal and various percentages everywhere else that would sum to 100% on the rows (actual class).

$$accuracy(y, \hat{y}) = \frac{1}{n_{samples}} \Sigma_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i) \tag{1}$$

$$confusion(i, j) = \frac{\Sigma(\hat{y}_i = y_j)}{\Sigma_{i=0}^{N} \Sigma(\hat{y}_i - y_j)} \tag{2}$$

We found that we had some moderate performers (i.e., LDA, Logistic Regression, QDA, Ensemble) and some stronger performers (i.e., KNN and SVM). We defined moderate performance as a score $\leq$ 80% and strong performance as a score $>$ 80%. Overall, all of our were able to perform much better than chance (~14%).

figures/KNN_params_wbg.png

Figure 11: KNN parameter space. More here

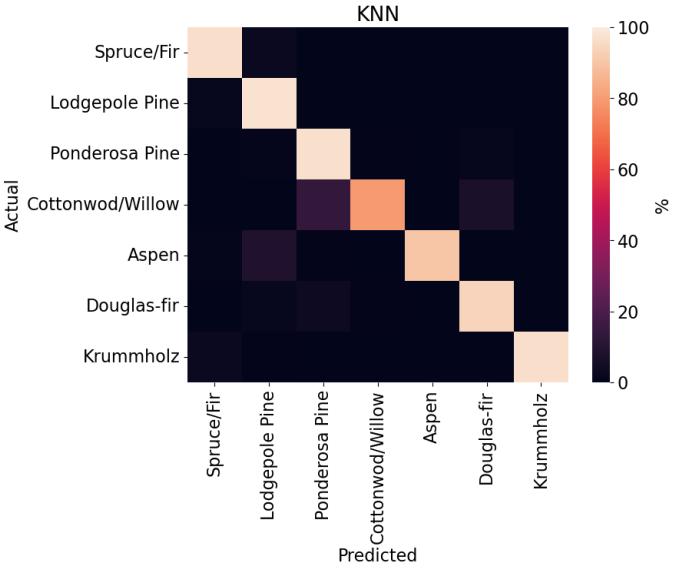

Figure 12: KNN confusion matrix. The x-axis contains the predicted classes and the y-axis contains the actual classes. The color of each element is determined by the relative accuracy of the predicted and actual combination to the overall number of instances for the actual class (Equation 2).

3.1 K nearest neighbors

We found that our KNN model achieved a accuracy of 97% (Figure 10). KNN outperformed each of our base models as well as our ensemble model. KNN was able to reliably predict a majority of the forest type covers; however, the model occasionally confused Cottonwood/Willow with Ponderosa Pine or Douglas-fir (Figure 12).

3.2 Support vector machine

Our SVM model received an accuracy of XX%(Figure 10). SVM was our second best model after KNN; however, upon further examination of SVM confusion matrix seen in (Figure 14) we can conclude that SVM strongly predicted three of seven classes. SVM poorly classified several classes, with Aspen being completely classified as Lodge pole Pine.

3.3 Linear Discriminant Analysis

We used 10 fold cross-validation to determine the best solver for a LDA model and found that the *svd* solver performed best (Figure 15). Our LDA model with the *svd* solver achieved an overall accuracy of ~67% (Figure 10). We found that our LDA model had an easier time predicting Spruce/Fir, Lodgepole Pine, and Krummholz than predicting Ponderosa Pine, Cottonwood/Willow, Aspen, and Douglas-fir (Figure 16). It appears that our LDA model seemed to confuse Ponderosa Pines with almost exclusively Douglas-firs and also confuse Cottonwood/Willows with Ponderosa Pines. The

figures/SVM_params_wbg.png

Figure 13: SVM parameter space. More here.

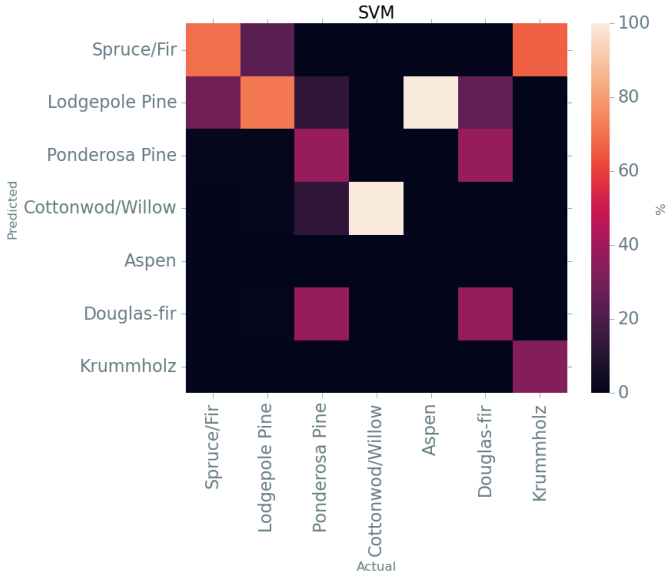

Figure 14: SVM confusion matrix. Predicted class on the x-axis and actual class on the y-axis. The color (and value) of the elements are found as in Table 2.

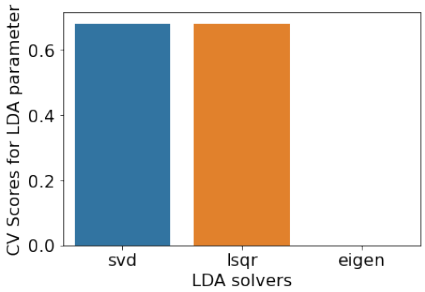

Figure 15: LDA parameter space. *svd* solver uses singular value decomposition; *lsqr* solver uses least squares solution; *eigen* solver uses eigenvalue decomposition.[7]

model also seems to have confused Douglas-firs for Pines (Ponderosa and Lodgepole). The model also seemed to confuse Aspens for exclusively Lodgepole Pines.

3.4 Quadratic Discriminant Analysis

We used 10 fold cross-validation to determine the best regulating parameter for a QDA model and found that a regulating parameter of 0.14 performed the best (Figure 17). Our QDA model with a regulating parameter of 0.14 achieved an overall accuracy of ~70% (Figure 10). We found that our QDA model had an easier time predicting Spruce/Fir, Lodgepole Pine, Ponderosa Pine, Cottonwood/Willow than predicting Aspen, Douglas-fir, and Krummholz (Figure 18). It appears that our QDA model mistook Aspens for Lodgepole Pines, Douglas-fir for Ponderosa Pines, and Krummholz for Spruce/Fir.

3.5 Logistic Regression

We used 10 fold cross-validation to determine the best parameters for a logistic regression model. We optimized over penalty (l1, l2, elastic net, none), C (regularization parameter), intercept term (True or False), and l1 ratio. We found that a logistic regression model with l2 penalty, C = 0.7525, no intercept term performed the best. Therefore, we created a logistic regression model using these parameters and achieved an overall accuracy of ~72% (Figure 10). We found that our logistic

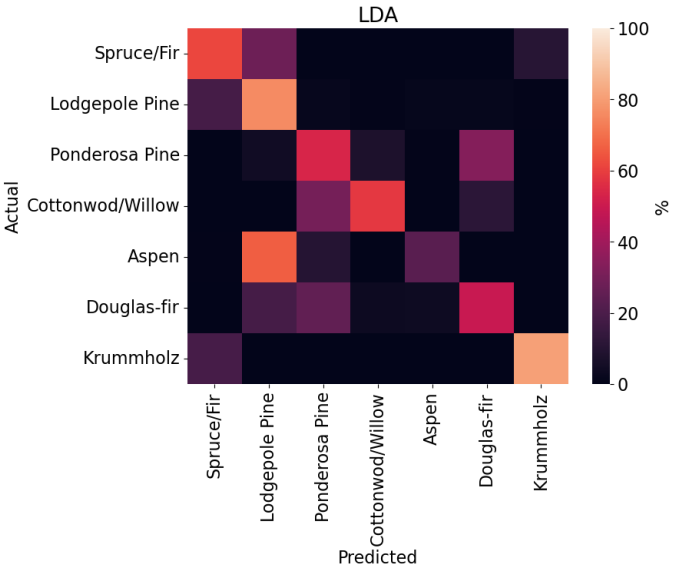

Figure 16: LDA confusion matrix. Predicted class on the x-axis and actual class on the y-axis. The color (and value) of the elements are found as in Table 2.

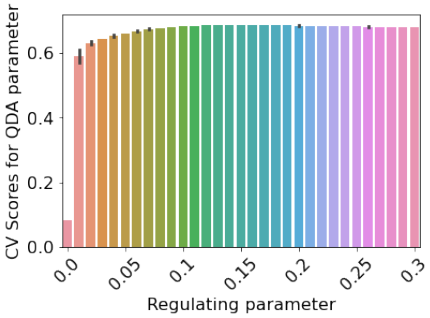

Figure 17: QDA parameter space. Regulating parameter refers to the regularization of per-class covariance [7]. Some bars have error bars due to binning of x-values and others do not have error bars because of no binning.

regression model classified Lodgepole and Ponderosa Pines well, but was not able to classify the other cover types as well. The logistic regression model confused Spruce/Fir and Aspen for Lodgepole Pine. It also confused Cottonwood/Willow for Ponderosa Pine. Douglas-fir was confused for both Pines (Lodgepole and Ponderosa), while Krummholz was confused for Spruce/Fir.

3.6 Ensemble

Our ensemble model received an accuracy of XX% (Figure 10).

4 Discussion

4.1 Moderate performers

Logistic Regression had the poorest predictive performance of our moderate performers. Logistic regression is a rather simple algorithm that tends to work best on binary classification problems due to the nature of the sigmoid curve used for classification. These reasons may provide an explanation for Logistic Regressions moderate performance; however, logistic regression also assumes there is little to no multicollinearity among features[9], which may occur in this data set.

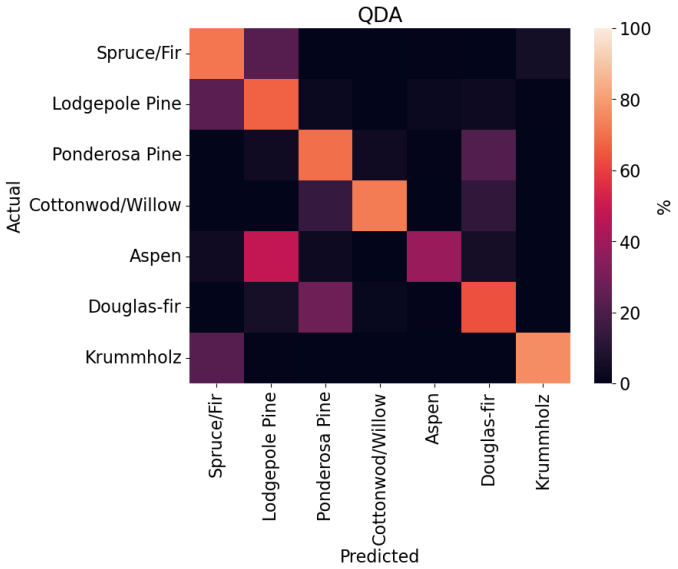

Figure 18: QDA confusion matrix. Predicted class on the x-axis and actual class on the y-axis. The color (and value) of the elements are found as in Table 2.

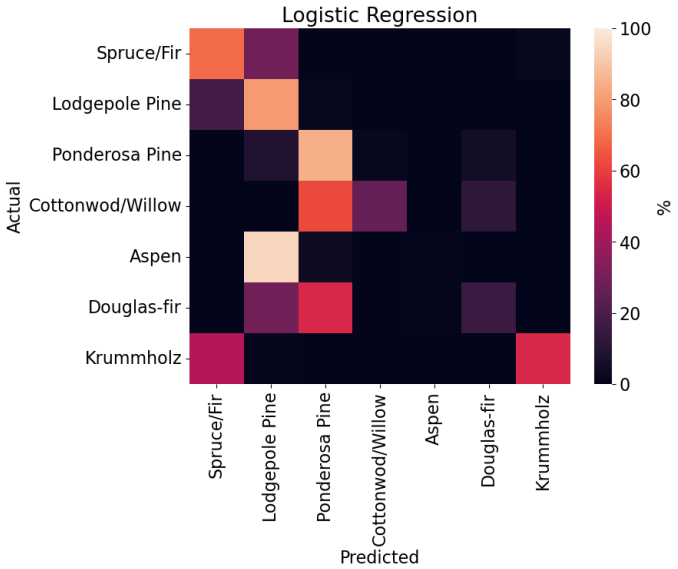

Figure 19: Logistic Regression confusion matrix. Predicted class on the x-axis and actual class on the y-axis. The color (and value) of the elements are found as in Table 2.

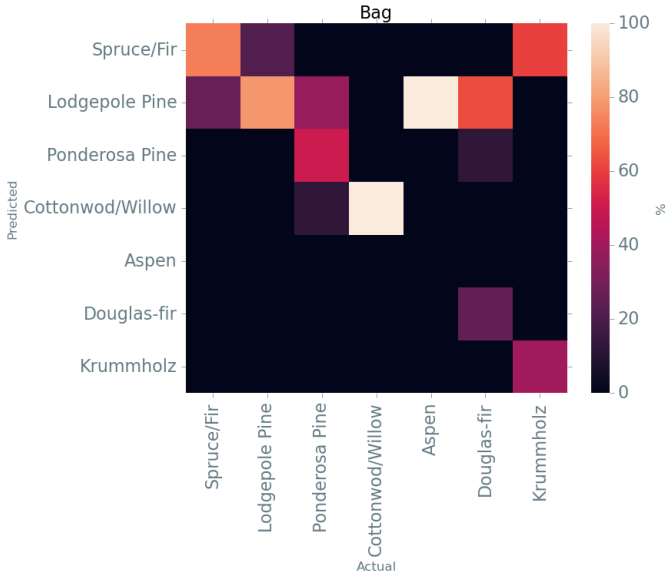

Figure 20: Ensemble model confusion matrix. Predicted class on the x-axis and actual class on the y-axis. The color (and value) of the elements are found as in Table 2.

LDA had the second poorest predictive performance of the moderate classifiers. As stated before, LDA assumes that each class shares the same covariance matrix. This assumption for the data set is unlikely to be true. This is evidenced by the performance of QDA who assumes to opposite of LDA; each class has it's own covariance matrix.

4.2 Strong performers

SVM had the second highest overall performance of the ensemble learners; however,

KNN performs well when n samples is far greater than n features [3]

SVM work best in higher dimensions (easier to establish "linear" hyperplane)

KNN vs SVM (SVM better when n features > n samples)

4.3 What happened with ensemble model?

One of the strengths of using an ensemble learning method with several types of models is that the varying assumptions underlying each model tend to work against each other. If LDA incorrectly assumes each class shares a covariance–which leads to multiple mislcassifications– than other models included in the ensemble that do not share this assumption will lead to an overall correct classification due to majority vote.

majority vote gone wrong. large amount Moderate classifiers.

ensemble performs better when all base models have similar accuracies

4.4 Future directions

1. introduce weights on each models predictions. base weights on a subset of the training data: a. base on accuracy of entire training set b. base on how accurate a certain model is a predicting a specific forest cover type. for example. KNN reliably identified most forest cover types well (figure 10 something); however, it had some trouble classifying Cottonwood/Willow. SVM, while reliably misclassifying Aspen as Lodgepole Pine, accurately classified Cottonwood/Willow.

2. mention long run time for SVM a. mention [4] whose used the Ripper Algorithm, similar to SVM, who initially used forest cover data set but did not obtain a score due to issues with run time.

3. Are moderate learners agreeing with each other? a. what about strong learners?
4. Adaboosting KNN

5 Replication

We provide all the code and data to replicate our results in a GitHub Repository. In order to replicate the analysis, one first needs to create an environment from the .yml file provided. We use, and highly recommend, Anaconda because of it's ease and ability to solve environments easily. To do this, open a terminal and type the following command:

```
conda env create -n <env-name> -f <env-yml-file>.yml
```

To replicate the figures from this paper, simply select the ENV environment and run both the NOTEBOOK notebook (simulated data) and NOTEBOOK notebook (analysis).

6 Acknowledgements

We would like to thank the authors of the data set: Jock A. Blackard, Denis J. Dean, and Charles W. Anderson.

7 Bibliography

References

[1] Jock A Blackard and Denis J Dean. "Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables". In: *Computers and electronics in agriculture* 24.3 (1999), pp. 131–151.

[2] Hristina Brynes and Grant Suneson. "The Largest and Smallest National Parks in America". In: *MSN* (2022). https://www.msn.com/en-us/travel/tripideas/the-largest-and-smallest-national-parks-in-america/ss-AAT2yFC.

[3] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. "An empirical evaluation of supervised learning in high dimensions". In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 96–103.

[4] Johannes Fürnkranz. "Round robin rule learning". In: *ICML*. Vol. 1. 2001, pp. 146–153.

[5] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[6] Nikunj C Oza and Stuart Russell. "Experimental comparisons of online and batch versions of bagging and boosting". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 2001, pp. 359–364.

[7] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[8] Omer Sagi and Lior Rokach. "Ensemble learning: A survey". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.4 (2018), e1249.

[9] NAMR Senaviratna, TMJA Cooray, et al. "Diagnosing multicollinearity of logistic regression model". In: *Asian Journal of Probability and Statistics* 5.2 (2019), pp. 1–9.