# Deep Feature Learning for (Random) Forest Cover Type Prediction

Jéssica Leal, Tim Kreienkamp and Philipp Schmidt

**Abstract:**
From a seven-class training data with 50,000 observations we try to find the best predictive model using different Machine Learning algorithm such as K-nearest neighbors, Support Vector Machine, Random Forest and Random Forest with deep learning. Our aim is to best predict this seven-class data set with 100,000 observations. For each model, we perform a 10-fold cross validation in order to assess the performance of our classifier in the training data set. After estimating the best hyperparameters for each model - with cross-validation - and comparing the train performance, we consider the Random Forest with deep feature learning is the best type of algorithm which can predict better our data.

## 1 INTRODUCTION

### 1.1 DATASET CHARACTERISTICS

In this project, we work with a training data set of 50,000 observations - with labels - and a testing set - without labels - of 100,000 observations. Our aim is to predict a seven-class variable in which each label is a different type of forest cover using cartographic variables. We have 10 numeric variables presented in a raw manner that needed to be scaled, and 44 binary variables for qualitative aspects of the forest such as the wilderness of that forest and the type of soil.

### 1.2 Baseline

Before considering a unique and a priori thought best classifier, we attempted to model our data with different other classifier such as the K-NN and the SVM in order to mark us a benchmark for future steps in classification.

## 2 Exploratory Analysis

## 3 Experiments

### 3.1 K-Nearest Neighbor

With this method, we try to classify each new observation taking into consideration the majority label from its k-nearest neighbor - in our case using the Euclidean distance. In order to see which hyperparameter k we should use, we use 10-fold cross-validation with different values of k, and we see that considering the label from the training set as a factor or as an integer does not affect our results. In Figure 3.1. and 3.2 we can see that the patter of the error in cross-validation for different values of the hyperparameter k is the same, and in both cases the best k to be chosen is 3. In the case in which Y is a factor, we get an error of 22.01%; and when Y is an integer we get an error of 22.066%. We consider the error is quite high in the training set, so this is one of the reasons why we decided to extend our models to take into account.

### 3.2 Support Vector Machine

We also try to approximate to predict the labels of this data set by separating the space into halfspaces with the largest margin possible.

### 3.3 Gradient Boosting

With this method we try to construct a classification tree of the labels why using one type of loss function: we use the multinomial distribution with a shrinkage parameter of 0.05.

### 3.4 Random Forest

We run a random forest in R with no additional parameter tuning on the full dataset. This gives us an accuracy of 0.80. From here on, we intend to make improvements.

### 3.5 Random Forest with deep learning

In a first set of experiments, we consider several well established classes of hypothesis functions. In particular, we consider Random Forests, Support Vector Machines, k-NN, Gradient Tree Boosting and Neural Nets (Deep Learning). We first run these over a grid of hyperparameters without any feature engineering. This gives us a feeling for how well different methods are suited for the task at hand. In grid search, one evaluates a classifier over a grid, i.e. the

cartesian product of several hyper parameter ranges. This process is computationally intensive, but since the data set is small, we can afford this luxury. The following figure shows the accuracy of the best model in each class. All accuracy values are obtained by a 10-fold cross validation procedure. For the deep learning models, we try Tanh and Rectifier activation functions, over 15 and 20 epochs. We try 3 and 4 hidden layers with each 200 hidden units and a no vs a small l1 regularization. The best model has 3 hidden layers, no regularization and uses a Tanh activation. For the random forest, we try 40,80,100, 150, 200 trees where 100 work best. For the gbm we consider interaction depths of 3, 5, and 7, and 40, 60 and 120 boosting iterations, where a depth of 7 and 120 boosting iterations work best. For the Knn, we consider neighbors up to 200, where 3 perform best.

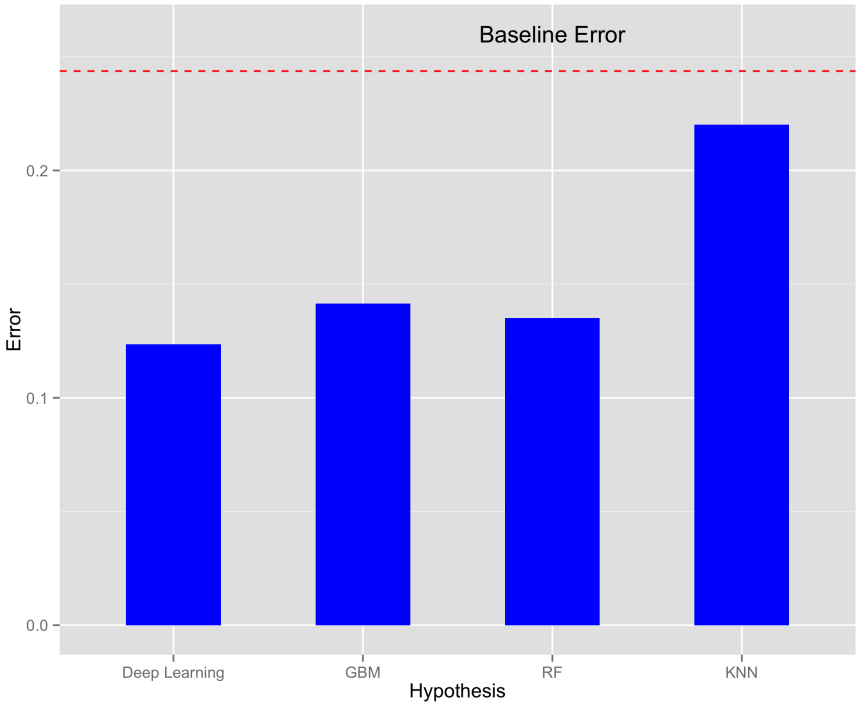


Figure 3.1: Initial errors

We clearly see that the performance of k-NN, at least in the tried configuration with this set of features is rather disappointing. Tree based methods and neural nets however, show - with some parameter tuning - a significant improvement over the baseline.

# 4 Feature Engineering and Model Tuning

## 4.1 Feature Learning

We settle on Deep Learning and Random Forests, since those performed best in the initial series of experiments. Based on our exploratory analysis, we add three new features to the

data set: Horizontal Distance to Hydrology (water), Vertical Distance to Hydrology, and Total Distance to Hydrology. We feed these into the our previously best performing models, which leaves us with a slight gain in accuracy. Since Deep Learning seems to perform well for this data set, we try a heuristic: Why not train the Random Forest on features extracted with Deep Learning? The recent success on Deep Learning seems to be partly based on it's ability to abstract high-level concepts from the data. For this reason, Machine Learning researchers use it for feature, or representation learning. Feature learning can be done in a supervised or unsupervised manner. We use it in a supervised manner based on the heuristic: the neural net with the highest accuracy will provide the best features for our random forest. Operatively, what is done is just that the nodes of the last hidden layer in the neural net are extracted and used to transform the original feature space. The following figure shows the accuracy of all methods described: Random Forests, Deep Learning, and Random Forests with "Deep Features".
We see that this results in a significant gain in accuracy. Apparently, Deep Learning is able to extract useful features from the data. However, in our last example we trained the random forest on 200 features. Too many features may potentially cause overfitting. This is why, in the next series of experiments, we add another layer to the neural net, where the size is significantly reduced, thus enforcing sparseness in the output. We try this with 50, 30 and 20 neurons.
Here we see, that we are able to again significantly reduce the error of the random forest.
Now we submit the models with 40 and 60 features for testing. Unfortunately, the test set error is significantly higher: namely we obtain an error of about 0.0965 for each method. There could be several reasons for this. First, the public leaderboard shows only a 10 percent portion of the actual test data. Second, and more likely is that we overfit: since the cross validation procedure is also only based on the training portion of the data, we might have captured it's structure too well. In fact, a highly complex method, like chaining deep learning and random forests could easily overfit. Therefore, we go back to random forests only and try to find a model that brings our cross validation error close to our training error.

### 4.2 Recursive Feature Elimination

Random Forests are able to give an estimate of the feature importances. We exploit this fact to use the random forest as a feature selection procedure. The procedure works as follows: We train a random forest and cross validate it. We compute the feature importances and the cross validation error. In the next iteration, we leave out the two least important features, and repeat the procedure, recording the CV-Error and the corresponding set of features each time. In the following graph, we see the results of said procedure.
We choose the best 15, 17, and 29 features (since these have a very low CV-Error), and see if we can tune the parameters further. We increase the fraction used at each iteration of the random forest, since this has been reported to increase accuracy on this dataset, to 0.95. We consider different values (20, 40 and 100) for the maximum depth each single tree has to be grown, where 40 and 100 perform equally well on each set of features. The following figure summarizes the errors for the best specification of the random forest algorithm on each subset.

We see that 17 features perform best. We try to increase the sample rate parameter which gives us a further improvement (to 0.075). Now we retrain on the full dataset and make predictions on the test set. The resulting submission scores 0.9319, which corresponds to an error of 0.0681. Since the test and and validation error are close in this case, and this model is our best in cross validation (apart from the apparently overfitting models built with deep learning), we decide to opt for this model as our final model. We think that it works best, because utilizes a well known method, and focuses on it's execution.

# 5 Limitations

# 6 Conclusions