Machine Learning Competition: Report on Forest Cover Type Prediction

Fernandez, María, McIver, Jordan, Besanson, Gaston

March 27, 2015

Introduction

THIS REPORT's purpose is to explain our experience on the Forest Cover Type Prediction's Competition.[1]. Our objective in this project was to *predict as best as possible the type of tree in a region with the purpose of minimizing the fires*. Even though we used the same loss for each type of misclassification -in other words, all trees are equally important-, we decided to create new features[2]. **This feature and others are explained in detailed in the attached PowerPoint presentation**.

[1] `https://inclass.kaggle.com/c/prediction-of-a-forest-covertype`

[2] The recommended approach would be consider penalizing more the misclassification of very flammable trees.

Data Exploration and Feature Creation

THE DATASET used in this report consists of 54 attributes (or features) and 50.000 observations for training (plus one class attribute) and 100.000 observations for the competition. The attributes are:

- *10 quantitative variables*

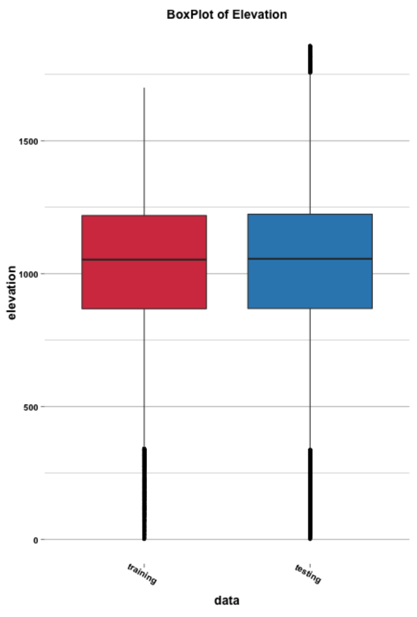
Feature	Range
Elevation	1859–3858 (m)
Aspect	0–360 (azimuth)
Slope	0–66
Horizontal Distance to Hydrology	0–1397 (m)
Vertical Distance to Hydrology	173–601 (m)
Horizontal Distance to Roadways	0–7117 (m)
Hillshade 9 a.m.	0–255 (index)
Hillshade Noon	0–255 (index)
Hillshade 3 p.m.	0–255 (index)
Horizontal Distance to Fire Points	0–7173 (m)

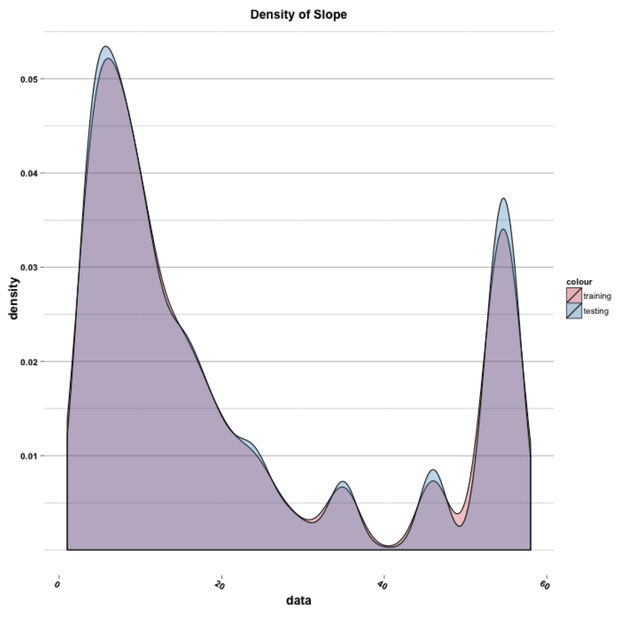
- *4 binary wilderness areas* and *40 binary soil type variables*
- *Cover Type*:

Class	Tree Type
1	Spruce/fir
2	Lodgepole pine
3	Ponderosa pine
4	Cottonwood/willow
5	Aspen
6	Douglas-fir
7	Krummholz

First, we quickly checked that our training set was representative of the testing set data. The means and quantiles of each feature where very similar between the two datasets. That gave us a hint, that the model training error would be a good proxy for the test error; *dismissing our initial concerns of overfitting*. The following boxplot shows the similarity of `Elevation` of the two datasets. A density plot of the `Slope` shows that the two datasets are equally distributed as well.[3]

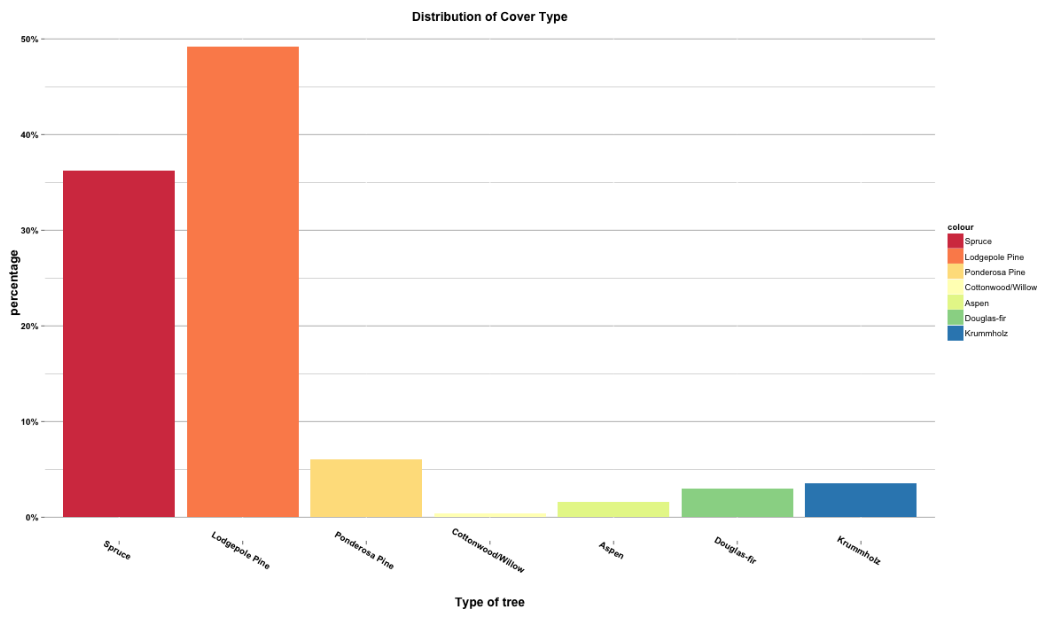
[3] For additional plots please see the presentation.





Second, we noticed that the training set is **imbalanced**[4]; meaning that we have two types of trees *Spruce* (Class 1) and *Lodgepole Pine* (Class 2) that represent over 80% of the observations.

[4] **This is also present in the original dataset `http://kdd.ics.uci.edu/databases/covertype/covertype.html`.**



WE CREATED six new variables to try to identify features important to fire risk.[5] Finally, we applied a *normalization* on both the training and the test sets to the 60 features.[6]

[5] we try, but did not continue, to reduce dimensionality through Principal Component Analysis, but the loss of variance decreased the models performance (there was loss of information). We tried linear and polynomial transformations of the training data and ran it with the Support Vector Machine method. Please see /Report/Graphs/old folder to see Plots and PCA code. Also, we decided not to collapse the binary variables. Please see: Nagel et al. *Exploring Non-Linear Data Relationships in VR using the 3D Visual Data Mining System.*

[6] `preProcess` function from the `Caret` package. We try also applying this pre-process to only the continuous variables, but did not make more difference.

New Feature	Type of Variable
No Sun at 3pm	Binary variable
Zero Horizontal Distance to Hydrology	Binary variable
Crow-fly Distance	Continuous variable
Fire Risk	Continuous variable
Elev. Shift-Vertical Distance to Hydrology	Continuous variable
Elev. Shift-Horizontal Distance to Hydrology	Continuous variable

For further detail see attached PowerPoint Presentation.

Machine Learning Methods

We are going to study the performance of following methods on the dataset.

K Nearest Neighbor[7]

[7] For this method, we used the `class` package

KNN is a lazy learning algorithms[8], that stores all available cases

[8] While a training dataset is required, it is used solely to populate a sample of the search space with instances whose class is known. No actual model or learning is performed during this phase.

and classifies new cases based on a similarity measure. We considered KNN because it doesn't need classes to be linearly separable and because it is very simple to implement with only two parameters to consider: the distance metric[9] and the neighbors being considered. On the down side, this algorithm is also sensitive to unbalanced dataset like this one; where infrequent classes are therefore often dominated in most neighborhoods.[10]

[9] Given that we used the build-in function of a package we were stuck with the Euclidean distance that works well with continuous variables but for categorical data other metrics can be considered.

[10] This can be alleviated through balanced sampling of the more popular classes, which we didn't do for this method

PARAMETERS TO TRAIN:

1. *k*: is the number of neighbors considered.

Support Vector Machine (SVM)[11]

[11] For this method, we used the `e1071` package

SVM tries to find a hyperplane that splits the data with the maximum margin. A clear shortcoming for plain-vanilla SVM is that this dataset is not linearly separable. To address this, we introduce the kernel transformations and try to classify the points. But, there is a second problem that the classes are imbalanced and this can produce suboptimal models which are biased towards the majority class and have low performance on the minority class, like most of the other classification paradigms. [12]. We included weights for the classes to try to overcome this problem; but the result was not promising because, even though it classified better the "smaller" classes, the training error was higher.

[12] Further readings, please see: Trebara,Mira,Steele, Nigel, *Application of distributed SVM architectures in classifying forest data cover types* Batuwita,Rukshan, Palade,Vasile, *Class Imbalance Learning Methods for Support Vector Machines*

PARAMETERS TO TRAIN:

1. *kernel*: nonlinear kernel function (such as a Gaussian radial basis function[13] or Polynomial kernel).
2. *cost*: is the parameter for the soft margin cost function, which controls the influence of each individual support vector; this process involves trading error penalty for stability.
3. *gamma*: is a kernel parameter.

[13] $K(x_i, x_j) = e^{-\frac{||x_i - x_j||^2}{2\sigma^2}}$, where σ is the spread or standard deviation.

Random Forest[14]

[14] For this method, we used the `caret` package

RANDOM FORESTS are an ensemble learning method that construct a number of decision trees at training time and outputs the class that is the mode of the classes output by individual trees. Random Forests are a combination of tree predictors where each tree depends on the values of a random vector sampled independently with the same distribution for all trees in the forest. The basic principle is that a group of "weak learners" can come together to form a "strong learner". *Random Forests are a wonderful tool for making predictions* *considering they do not overfit because of the law of large numbers.*[15]. An important characteristic of this method is that it provides methods for balancing error in class population unbalanced datasets.

[15] Further readings, please see: Sug,Hyontai, *Better Induction Models for Classification of Forest Cover*

PARAMETERS TO TRAIN:

1. *Randomly Selected Predictors*: is the number of attributes to pick randomly to generate each subtree in a tree in the forest. The recommended default parameter value is the square root of the number of parameters.
2. *Number of trees*: The recommended default parameter value is among the hundreds.

Gradient Boosting Machine (GBM)[16]

[16] For this method, we used the `H2o` package. We tried first with the `gbm` package, but found it was unstable. **Please remember that the `H2o` package runs on `Java Virtual Machine`, it uses the `RJava` package. If you are using the AWS machine with the Rstudio image you have to to install `JVM` and `RJava` from the command line**

ON THE CONTRARY to Random Forest, that relies on simple averaging of models in the ensemble; the main idea of boosting is to add new models to the ensemble sequentially. At each particular iteration, a new weak, base-learner model is trained with respect to the error of the whole ensemble learnt so far. To establish a connection with the statistical framework, a gradient-descent based formulation of boosting methods was derived. This formulation was called the gradient boosting machines.

IN GBMs, the learning procedure consecutively fits new models to provide a more accurate estimate of the response variable. The principle idea behind this algorithm is to construct the new base-learners to be maximally correlated with the negative gradient of the loss function, associated with the whole ensemble.[17] We picked this method because it introduces a lot of freedom into the model design.

[17] Further readings, please see: Natekin,Alexey, Knoll, Alois, *Gradient boosting machines, a tutorial*

PARAMETERS TO TRAIN:

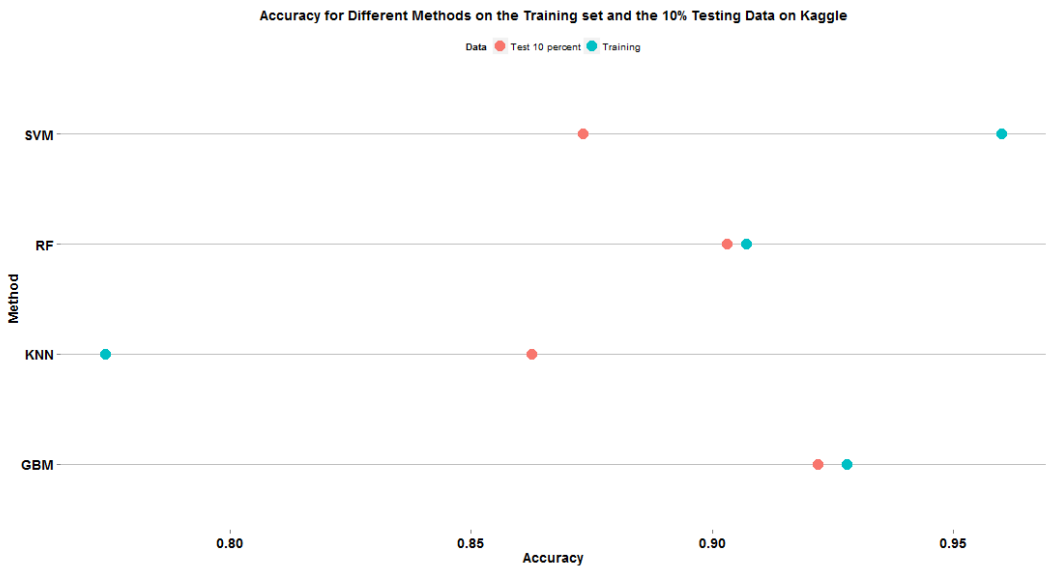
1. *Number of trees*.
2. *Maximum Depth*: The maximum number of edges to generate between the first node and the terminal node.
3. *Minimum Rows*: The minimum number of observations to include in a terminal leaf.
4. *Shrinkage*: The rate at which the algorithm should converge.

Results

FOR EACH METHOD we tried different parameters, especially with the ensemble methods which where our top picks for this competition.

To pick among these parameters, we did 10-fold cross validation. Given the amount of parameters that we tried for Random Forest and GBM, the training of the models was done in batches. **Please see in the Appendix the detailed information on training the models for the different methods**.

THE FOLLOWING plot shows the best model for each method and the corresponding accuracy in the training set proxied by the 10% sample leaderboard from the Kaggle Competition.



Conclusion

CONSIDERING our results on the training set and acknowledging the results on the 10% of the Kaggle Competition, our final submission for this competition is the GBM model with the following parameters:

1. *Number of trees*: 250
2. *Maximum Depth*: 18
3. *Minimum Rows*: 10
4. *Shrinkage*: 0.1

FINALLY, we want to show the Confusion matrix for this model:

```
|Predicted|
```

Actual	1	2	3	4	5	6	7	Error
1	16523	1501	1	0	11	0	65	0.08718
2	1016	23382	59	0	71	47	16	0.04916
3	0	113	2811	15	4	83	0	0.07105
4	0	1	52	131	0	16	0	0.34500
5	19	250	16	0	520	5	0	0.35802
6	4	125	198	10	2	1172	0	0.22435
7	177	18	0	0	0	0	1566	0.11073
Totals	17739	25390	3137	156	608	1323	1647	0.07790

The larger misclassification happens between Class 1 and Class 2 as we mention in the start of this Report, it accounts for 66% of the misclassification between Classes.

Appendix

KNN

K	Accuracy	Public Score (10%)
1	**0.77418**	**0.8626**
2	0.74394	
3	0.76902	

SVM

Cost	Gamma	Accuracy	Public Score (10%)
0.1	0.5	0.77244	
1	0.5	0.89456	
10	**0.5**	**0.96026**	**0.87320**
0.1	1	0.71196	
1	1	0.92776	
10	1	0.98742	0.78030
0.1	2	0.56352	
1	2	0.96114	
10	2	0.998	0.79920

Random Forest

At first we used 3 and 5 folds Cross Validation to discard some of parameters.

mtry	Accuracy	CV	Public Score (10%)
8	0.8548800	3 Folds	
16	0.8802400	3 Folds	
24	0.8843601	3 Folds	
32	0.8862000	3 Folds	
40	0.8865000	3 Folds	
48	0.8868199	3 Folds	

mtry	Accuracy	CV	Public Score (10%)
56	0.8872399	3 Folds	
59	0.8867800	3 Folds	
8	0.8620399	5 Folds	
16	0.8894999	5 Folds	
24	0.8926399	5 Folds	
32	0.8953000	5 Folds	
40	0.8958948	5 Folds	
48	0.8992497	5 Folds	
56	0.8970799	5 Folds	
59	0.8956799	5 Folds	
48	**0.9030799**	10 Folds	**0.90770**
56	0.9017999	10 Folds	

GBM

Initially we left shrinkage (equal to 0.1) and the Minimum Rows (equal to 10) parameters untouched. But when we found the best GBM model we try then tried a grid for different Minimum Rows: 1,5,7.

Depth	Trees	Accuracy	CV	Public Score (10%)
8	100	0.87672	3 Fold	
10	100	0.89244	3 Fold	
12	100	0.90042	3 Fold	
14	100	0.90382	3 Fold	
10	250	0.89106	3 Fold	
12	250	0.90648	3 Fold	
14	250	0.90692	3 Fold	
16	250	0.90904	3 Fold	
14	500	0.90506	3 Fold	
16	500	0.90652	3 Fold	
14	250	0.9145	5 Fold	

Depth	Trees	Accuracy	CV	Public Score (10%)
16	250	0.91694	5 Fold	
14	500	0.9124	5 Fold	
16	500	0.91258	5 Fold	
16	250	0.91872	10 Fold	
18	**250**	**0.9221**	**10 Fold**	**0.92790**
22	250	0.92094	10 Fold	