

Machine Learning: Kaggle competition on Prediction of a forest coertype

Samantha Dalton, Yevgen Levin, and Jordi Zamora Munt

26/03/2015

Goal

Use a training data set of 53 features and 50000 observations to predict from a testing set of 100000 observations 7 different forest coertypes. Using the Kaggle competition platform we will evaluate the performance of the method by measuring the accuracy on the testing set.

Previous works have reached up to 3% of error rate with the whole data set that contains more than 500k examples by using random forests (<http://www.wise.io/blog/benchmarking-random-forest-part-1>).

Introduction of the data

The data provided is divided in a training set and a testing set. The training set contains 50000 examples with 53 features plus the corresponding id's and coertypes. The structure of the features is as follows:

- Name / Data Type / Measurement / Description
 - id
 - Elevation / quantitative / meters / Elevation in meters
 - Aspect / quantitative / azimuth / Aspect in degrees azimuth
 - Slope / quantitative / degrees / Slope in degrees
 - Horizontal-Distance-To-Hydrology / quantitative / meters / Horz Dist to nearest surface water features
 - Vertical-Distance-To-Hydrology / quantitative / meters / Vert Dist to nearest surface water features
 - Horizontal-Distance-To-Roadways / quantitative / meters / Horz Dist to nearest roadway
 - Hillshade-9am / quantitative / 0 to 255 index / Hillshade index at 9am, summer solstice
 - Hillshade-Noon / quantitative / 0 to 255 index / Hillshade index at noon, summer solstice
 - Hillshade-3pm / quantitative / 0 to 255 index / Hillshade index at 3pm, summer solstice
 - Horizontal-Distance-To-Fire-Points / quantitative / meters / Horz Dist to nearest wildfire ignition points
 - Wilderness-Area (4 binary columns) / qualitative / 0 (absence) or 1 (presence) / Wilderness area designation
 - Soil-Type (40 binary columns) / qualitative / 0 (absence) or 1 (presence) / Soil Type designation
 - Cover-Type (7 types) / integer / 1 to 7 / Forest Cover Type designation

The testing data contains the same features except for the Cover-Type that is missed and it is what we want to predict.

Visual exploration of the training data

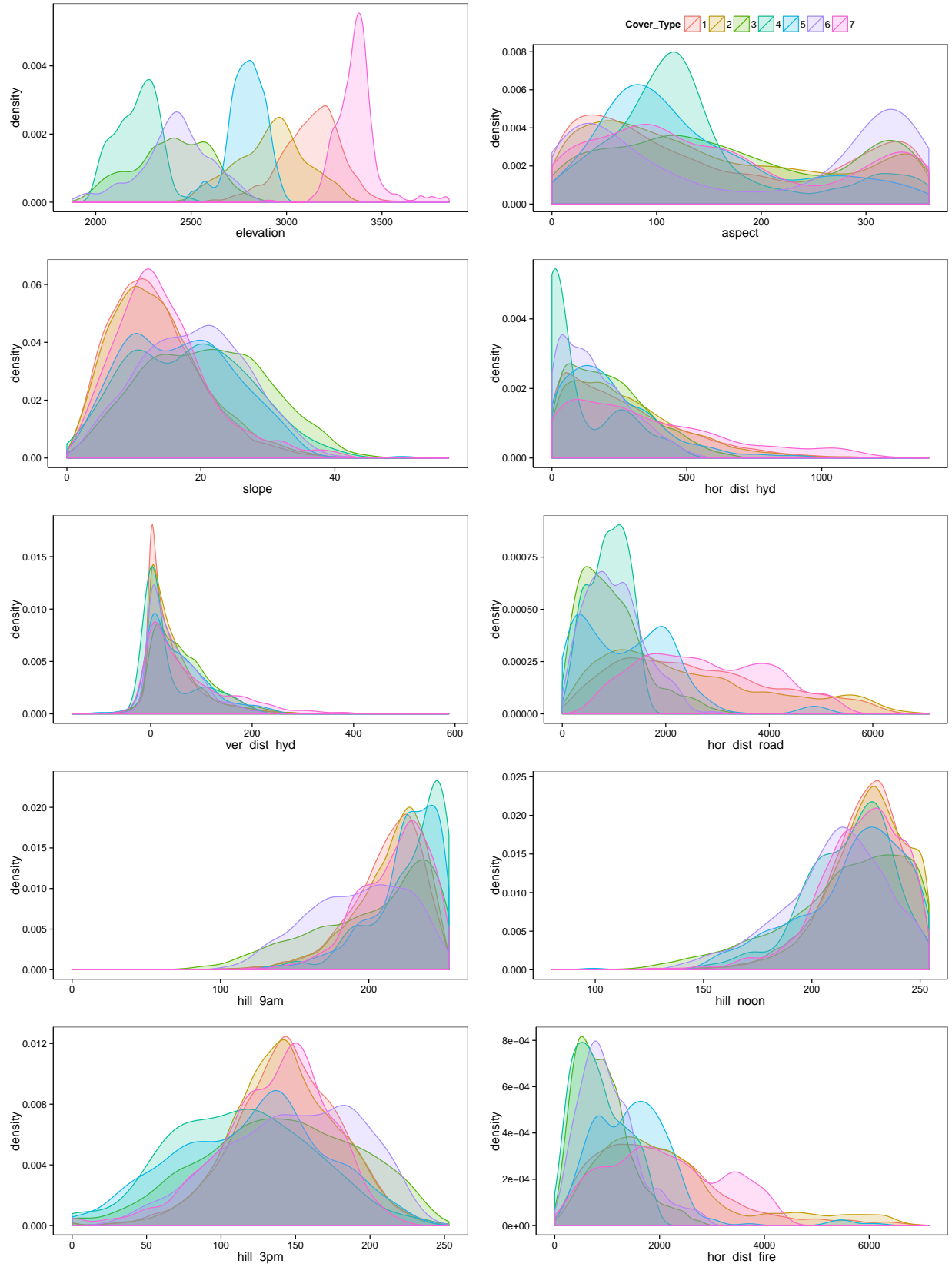
The first data check we should do is look at the distribution of Cover_Type in the data so we know if we are dealing with equal sample sizes across each cover type.

```
## Warning: package 'ggplot2' was built under R version 3.1.3
```

```
##
##      1      2      3      4      5      6      7
## 18101 24591 3026   200   810  1511  1761
```

The table shows that covertypes 1 and 2 are much more prevalent than other types in the training data. Covertypes 4 and 5 are poorly represented in the training set which can lead to difficulties when we will try to train a model to predict these underrepresented groups.

Another way to understand the data before modeling it is to do some simple visualization. To view variation among covertypes across the continuous variables in the data we created density plots of these variables colored by covertime.



The features that seem to capture display the most variation among covertypes seems to be captured by

elevation and aspect. The rest of the features show a strong overlapping of the covertypes.

For the binary features, we show a table that summarizes the 4 wild areas and the 40 soil types.

##		Perc	counts	Cov_1	Cov_2	Cov_3	Cov_4	Cov_5	Cov_6	Cov_7
##	wild_area_1	0.449	22438	8964	12723	0	0	308	0	443
##	wild_area_2	0.051	2526	1592	743	0	0	0	0	191
##	wild_area_3	0.438	21901	7545	10869	1220	0	502	638	1127
##	wild_area_4	0.063	3135	0	256	1806	200	0	873	0
##	soil_type_1	0.005	250	0	0	171	11	0	68	0
##	soil_type_2	0.013	649	0	77	430	6	21	115	0
##	soil_type_3	0.008	405	0	101	211	72	0	21	0
##	soil_type_4	0.022	1083	12	309	638	15	50	50	9
##	soil_type_5	0.003	134	0	0	81	4	0	49	0
##	soil_type_6	0.011	541	0	68	337	18	0	118	0
##	soil_type_7	0.000	9	0	9	0	0	0	0	0
##	soil_type_8	0.000	22	3	19	0	0	0	0	0
##	soil_type_9	0.002	108	14	94	0	0	0	0	0
##	soil_type_10	0.057	2834	74	965	982	18	20	775	0
##	soil_type_11	0.021	1064	72	765	108	0	67	52	0
##	soil_type_12	0.053	2639	214	2425	0	0	0	0	0
##	soil_type_13	0.030	1493	167	1137	1	0	133	55	0
##	soil_type_14	0.001	46	0	0	7	15	0	24	0
##	soil_type_15	0.000	0	0	0	0	0	0	0	0
##	soil_type_16	0.005	249	61	156	2	3	5	22	0
##	soil_type_17	0.006	318	23	88	49	38	56	64	0
##	soil_type_18	0.003	166	6	147	0	0	13	0	0
##	soil_type_19	0.007	374	234	135	0	0	5	0	0
##	soil_type_20	0.016	780	301	457	0	0	7	15	0
##	soil_type_21	0.002	76	73	2	0	0	0	0	1
##	soil_type_22	0.057	2856	2216	626	0	0	0	0	14
##	soil_type_23	0.099	4954	3119	1726	0	0	44	6	59
##	soil_type_24	0.036	1819	950	834	0	0	5	8	22
##	soil_type_25	0.001	37	16	21	0	0	0	0	0
##	soil_type_26	0.005	236	23	201	0	0	12	0	0
##	soil_type_27	0.002	86	46	39	0	0	0	0	1
##	soil_type_28	0.002	82	4	76	0	0	2	0	0
##	soil_type_29	0.199	9963	3567	6242	0	0	88	0	66
##	soil_type_30	0.052	2591	629	1760	0	0	181	0	21
##	soil_type_31	0.044	2198	1017	1138	0	0	26	2	15
##	soil_type_32	0.088	4420	1791	2497	8	0	41	13	70
##	soil_type_33	0.078	3882	1529	2206	1	0	33	53	60
##	soil_type_34	0.003	143	8	130	0	0	1	1	3
##	soil_type_35	0.004	197	95	0	0	0	0	0	102
##	soil_type_36	0.000	8	1	0	0	0	0	0	7
##	soil_type_37	0.001	26	0	0	0	0	0	0	26
##	soil_type_38	0.026	1320	723	74	0	0	0	0	523
##	soil_type_39	0.024	1193	695	35	0	0	0	0	463
##	soil_type_40	0.015	749	418	32	0	0	0	0	299

The first thing we notice is that there are no covertypes with nonzero values for soiltype 15. It means this feature has no predictive power and we should remove it from the feature list. A second important observation is that examples of covers 1 and 2 are both desbriced by similar counts of soils and wild areas. Finally, soiltypes 7 and 37 only contain observations with covers of type 2 and 7 respectively which can be an unique identifier of class type if an observation has one of these soiltypes indicated.

Proposed methods

We have done some data preprocessing using different approaches.

- Raw data inspection: From the previous section we removed soiltype 15 since we do not expect any predictive power from it.
- Data scaling: We evaluated the accuracy of the different methods by standardizing the data. Three different cases were compared: Scaling all features, Scaling the continuous valued features, weighting the binary features.
- Dimension reduction methods: Together with the previous scaling we used the R packages ‘princomp’ for principal component analysis and ‘svd’ for singular value decomposition. We wanted to apply the classification algorithms in a lower dimensional space and remove redundant factors.

We have focused our classification efforts on three classification methods: Random Forest, Support vector machines and k-Nearest Neighbours.

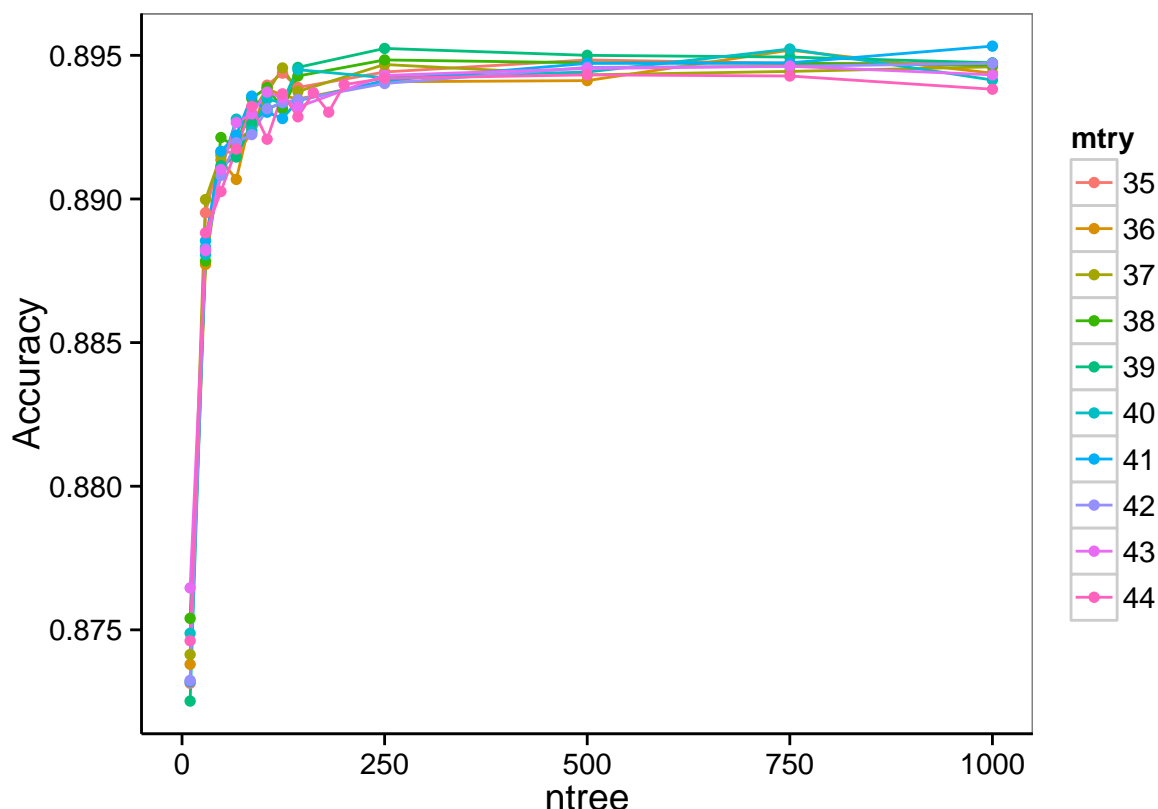
This first method we tried for classification was Random Forest. This is a popular ensemble technique that relies on selecting random subsets of the features and random subsamples of the data to partition the features space with axis parallel cuts (trees). It generates a set of partitions each of which is limited during the division process and is labeled by majority vote among the data points that are in that partition. We used the `randomForest` package that implements Breiman’s random forest algorithm (see http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm for further details). Because Random Forest is quite popular for classification problems, it seemed appropriate to pursue this method for this problem.

The second approach we attempted for classification was Support Vector Machines. SVMs are used fairly often in supervised learning. Thus they seemed like a viable option for classification. Also since the data did not appear to be linearly separable at first glance, SVMs are appealing because they can efficiently perform nonlinear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

The final approach we tried and we used in our final submission (with a more sophisticated implementation using weighted binary feature) was the k-Nearest Neighbors algorithm. kNN is a simple algorithm that calculates distance from a point to its k nearest neighbors. It assigns the point to the same class as the majority class of its k nearest neighbors. The class package in R only allows for distance of features to be measured using Euclidian distance, which can be a limitation.

randomForest

Raw random forest: The best results using random forest were obtained by using all the features as provided in the original data. Using the function `tuneRF` we were able to do a coarse grain selection of the number of features randomly chosen on each iteration of the method (*mtry* parameter). The optimal *mtry* was around 40 that is far from the *mtry*=7, that is the default value, due to the sparsity of the binary data. In a second step we perform a more systematic optimization around this value. In this process we cross-validated the accuracy with 8 buckets out-of-sample running the code in AWS with 8 cores. The results are shown in the figure below.



The accuracy grow assymptotically to ~ 89.5 for all the *mtrys*. The optimum accuracy of 0.8953 is obtained for *mtry*=41 and *ntree*=1000. However, in Kaggle we decided to submit our second best guess that was for *mtry*=39 and *ntree*=1000 and it returned a decent 0.90090 of accuracy.

We also tried other approaches that are summarized in the following table:

Method	Preprocessing	Accuracy(%)	mtry	ntree	Other Params
Random Forest	raw data	89.47	39	1000	
Random Forest	PCA without scaling	88.14	12	50	depth=all
Random Forest	PCA with scaling	87.69	34	50	depth=all
Random Forest	SVD without scaling	89.46	37	124	depth=6

There were two reason to try SVD and PCA in the preprocessing of the data. The first one was to project the data in a more suitable base such that the random forest would require less trees to classify the data. The second was to get rid of unnecessary or redundant features. The depth of the dimension reduction is given by the *depth* parameter. For PCA the best results were obtained by including all the features while for SVD we observed that the optimal depth was for the first 6 components, reducing the computation time significantly. Those results are somehow reasonable since the random sample of features in the random forests are thought to reduce the effect of noisy features. In a last trial, not reported, we used the relative importance of the features from the output of the randomForest method to get rid of the less important features. Unfortunately, dimension reduction returned reduced accuracy in crossvalidation.

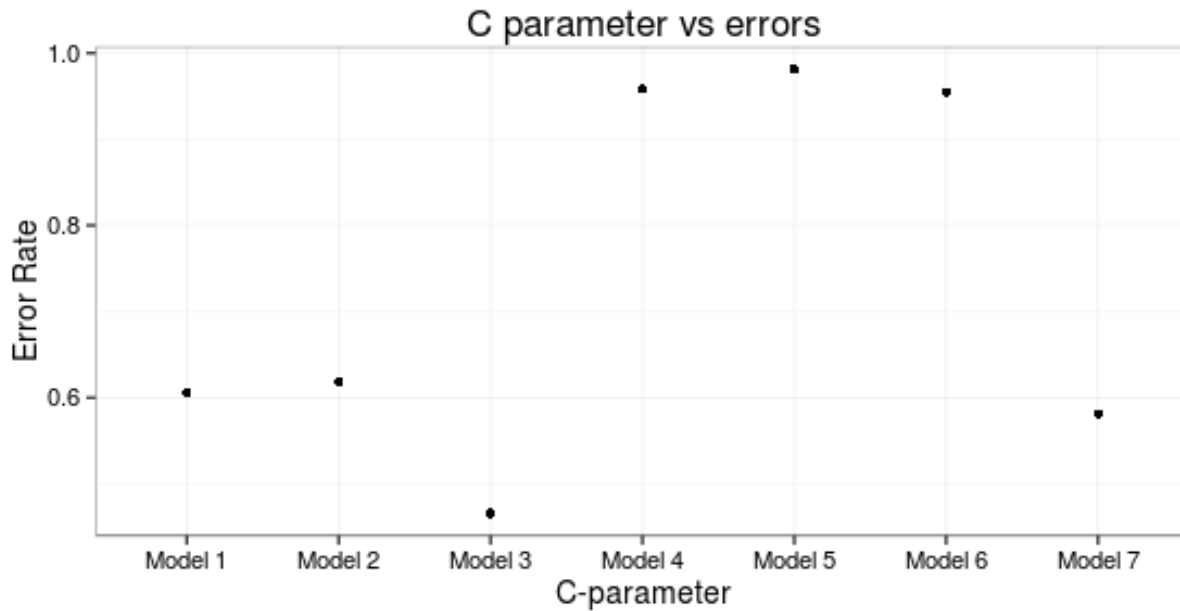
Support Vector Machines

We first tried using SVMs to classify one group against all others, but this method did not perform nearly as well as the kNN and randomForest methods. Given the enormous amount of computing time SVMs require, we explored other SVM models and packages in R and came across specifications that are adaptable to multiclass problems.

A brief note should be made about the methodology to find the ideal parameters in the SVM models considered. The initial approach was to estimate and perform diagnostics on the model trained on the entire training set. However, the computation time was over 4 hours in initial trials, so we adjusted our strategy and tuned the SVM parameters on only 20% of the training set provided. This approach has allowed a greater set of models to be tested while still maintaining the relative differences in performance across models.

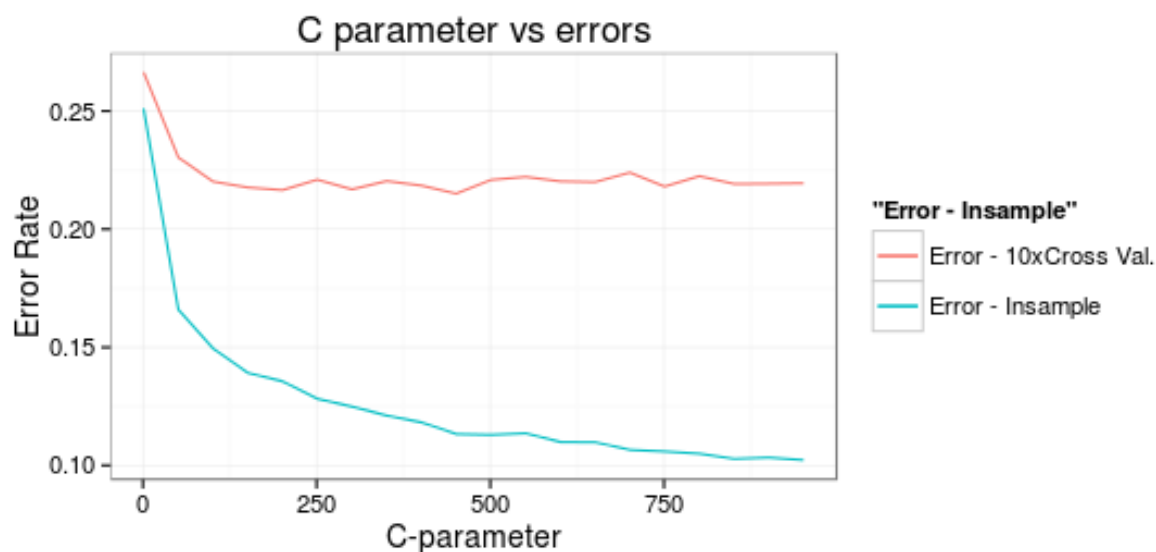
All of the types of SVM models considered were run in two different versions, considering the transformation of the continuous variables. These were scaled according to subtracting the mean and dividing by standard deviation of the training sample. The rationale is that this would make the numerical optimisation routine more stable. Some experimentation with different kernels was done in the training of the SVM. The performing kernels were the RBF and the Laplace kernel; the other kernels were severely underperforming in comparison.

One-against-all SVM Initial set of models tested considered was the one against all SVM classification for each class. So given 7 classes, 7 different model were run. The performance of these is evaluated using both in-sample validation error and cross validation error. The results are presented graphically below (for both scaled and unscaled features). Error measures were incredibly high and thus these models were discarded instantly.



Multi-class SVM Since this is a multiclass classification problem, we decided to try a classic approach called the ‘one-against-one’ method where we train $k(k-1)/2$ binary classifiers. Here, k represents the number of classes (7 in our case). The appropriate class is determined using a voting scheme. Once again, the RBF kernel was used for training, as experimentation with other kernels yielded inferior error rates (scaling the continuous variables provided miniscule improvement for this method).

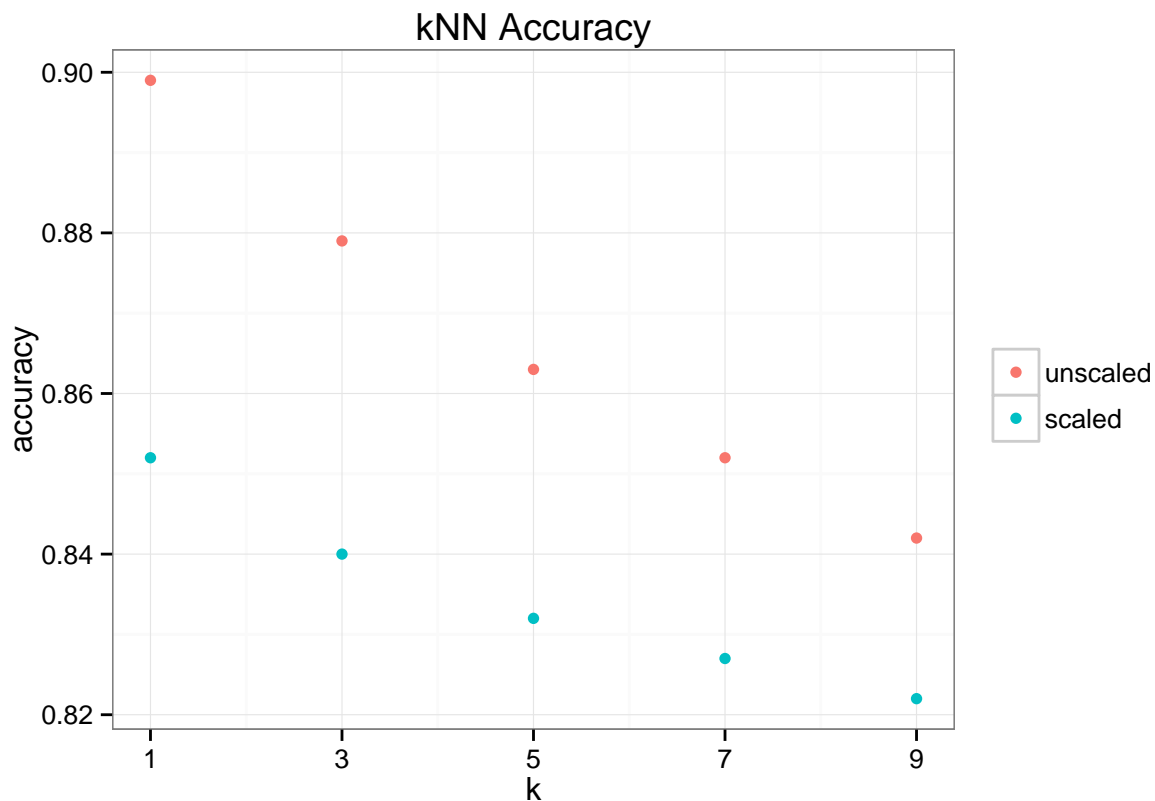
We found that increasing the regularization C parameter increased in-sample training accuracy significantly. The relationship of the C parameter and the error rates can be seen in the graph below. However, increasing the C parameter also greatly increases training time. This is because an increased C parameter assigns a higher penalty to misclassification and forces the SVM optimisation routine to search more look for a “harder” boundary. From the below graph we can see that marginal improvement from increasing C decreases greatly as C increases. We chose arbitrarily $C=5000$ as a reasonable parameter in accuracy/run time pay off. The final in sample misclassification error was 0.1072. The 10 fold cross validation however was much greater.



SVM models proved to be inefficient with regard to run time and accuracy when compared to other models we considered. Thus we discarded this class of models as suitable for the given problem.

Nearest Neighbours

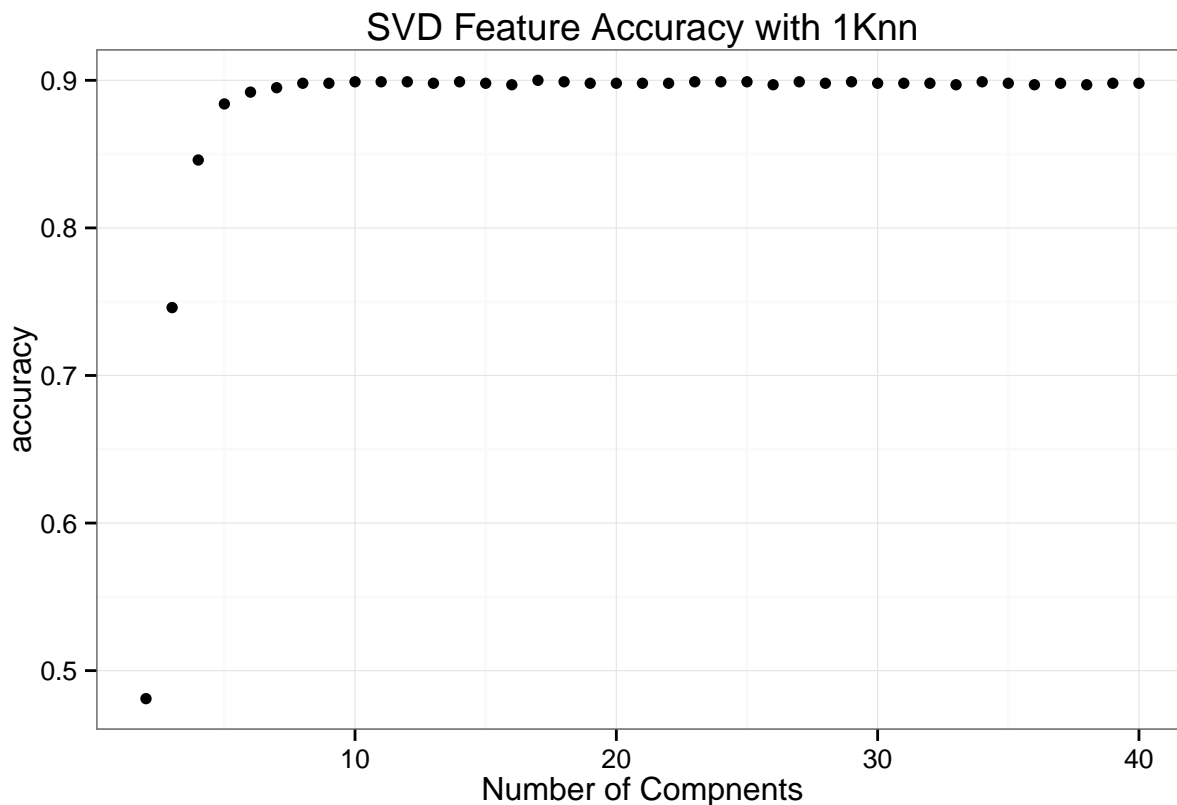
The nearest neighbor algorithm gave some promising results as a classifier. We tried to optimize the k for the nearest neighbor algorithm on both the raw data and the data that centered and scaled the continuous features. For both instances, the highest accuracy was found using $k=1$. Surprisingly, scaling the continuous features to be much smaller and centered around 0 did not increase the predictability of this classifier. So we decided to use the raw data in future iterations of this algorithm. Below is a chart displaying the accuracy of the kNN algorithms across k 's for the scaled and unscaled data.



Another technique we tried was using singular value decomposition (SVD) with the 1NN algorithm. The SVD can be thought of as decomposing a matrix into a weighted, ordered sum of separable matrices. The idea was that perhaps dimension reduction that focused on the most important orthogonal features would give us a better measure of distance between points. While this method did perform well; it did not improve accuracy much more than 1NN. Below is a chart displaying the optimal number of SVD components to use with the 1NN algorithm. The optimal number of components to use (the one with lowest error) was 16, but as you can see from the chart, accuracy does not increase much after including 10 components (which is still quite a large reduction in dimension).

```
knn_svd <- read.csv("knn_svd_errors.csv", header=T)
knn_svd1<-aggregate(knn_svd$error, list(knn_svd$features),mean)
max<-c(rep(1,length(knn_svd1$x)))
knn_svd<-data.frame(no_components=knn_svd1$Group.1, accuracy=round(max-knn_svd1$x,3))

ggplot(data=knn_svd,aes(x=no_components,y=accuracy)) + geom_point() + ggtitle("SVD Feature Accuracy with")
  xlab("Number of Compnents")
```



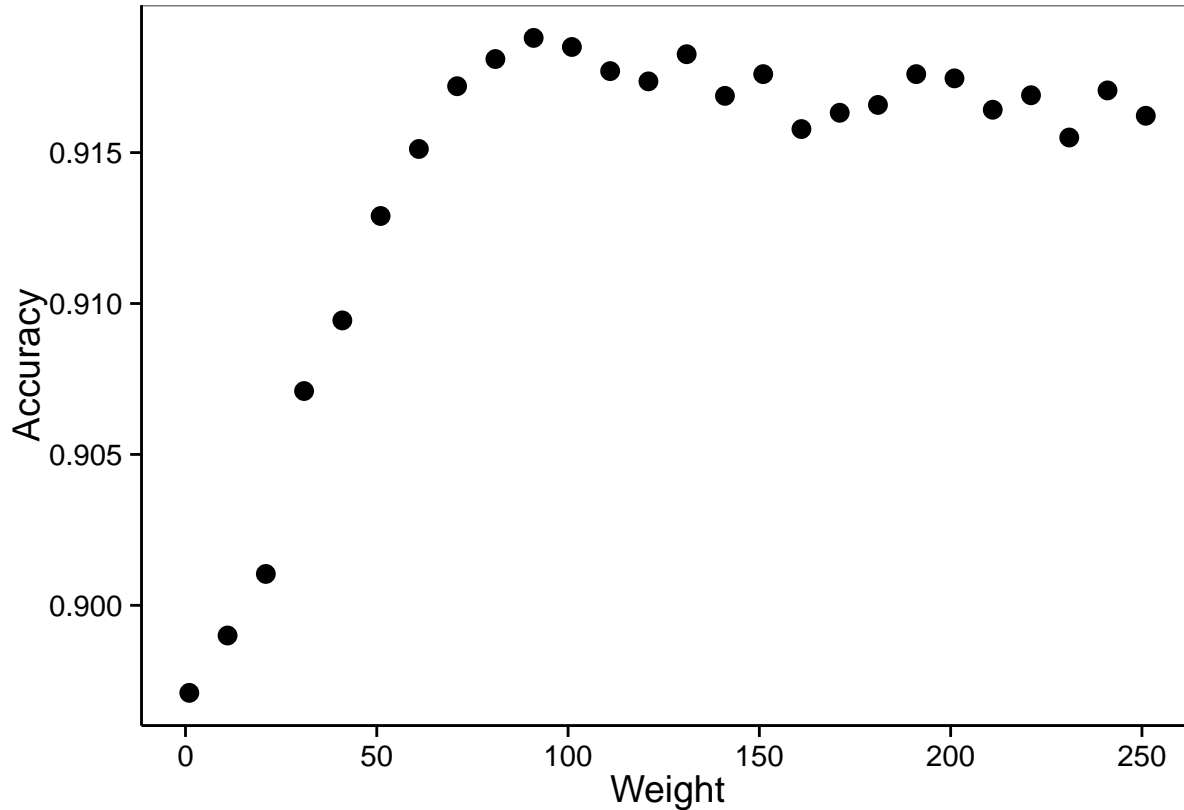
Best Classifier: Nearest Neighbours with weighted binary data

In order to improve the accuracy we realized that the wild areas and soil types were underweighted in the calculation of the distance. This is because the scales of the features are substantially different. For example, the elevation ranges from 2000 to 6000 while the soil types and wild areas are binary, i. e. they can take values 0 or 1. The binary differences lead to an insignificant increase in the distance when kNN is comparing examples with and without a given binary feature. So the first thing we tried is to scale the continuous variables. The results are (ADD DOCUMENTATION ON SCALED KNN)

To solve this issue we came back to the visual inspection of the data. In the calculation of the distance we want to keep the relevance of the continuous features since some of them are reasonably well separated (e.g. the elevation) and increase the relevance of the binary features. To do that we multiplied the binary features by the same weight while keeping the continuous variables with the default scale. The optimal parameters we got were $weight = 91$ and $k = 1$. The figure below shows the optimization of the weights for $k = 1$.

```
kNN_OptimalWeight <- read.csv("1NN_OptimalWeight.csv")
names(kNN_OptimalWeight) <- c("Misclass", "Weight")
ggplot(data=kNN_OptimalWeight,
       aes(x=Weight,y=(1-Misclass),size=2)) +
  geom_point() +
  ylab(c("Accuracy")) +
  theme_bw() +
  theme(axis.line = element_line(colour = "black"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        text = element_text(size=14),
```

```
legend.position="none")
```



We also checked systematically that the best k was 1 for the different weights. The optimal results for other weighting and scaling strategies we checked are summarized in the following table

Method	Preprocessing	Accuracy(%)	k	Weight	Other Params
kNN	raw data	91.88	1	-	
kNN	Scaled continuous		1	N(0,1)	
kNN	Weighted binaries		1	91	
kNN	SVD		1		

Conclusions

We conclude that the best method we could find to classify the forest covertypes was the kNN with $k=1$ with a weight of 91 applied to the binary variables, so values were $\in \{0, 91\}$. This method is responsible for our best Kaggle submission: an accuracy rate of 92.52% in the 10% test set. Our approach relies on the fact that the binary variables are underweighted in the calculation of the euclidean distances used in kNN. Since the scaling of the continuous variables didn't give us improved results we decided to scale all the binary variables with a fixed weight. This method gives the best compromise between the relevance of the continuous features and the binary features. It is possible with more time and exploration that weights could vary across for each individual binary feature, but this was not something we explored.

We discarded many methods in our search for the optimal classifier; many performed only slightly worse than our final classifier. The PCA preprocessing was the most disappointing; it probably underperformed because it is designed to reduce dimensionality and capture as much variance as possible in very few variables. While this method did manage to do that, the dimensions that it excluded probably contained enough accuracy to provide one or two extra points of accuracy in other methods. SVMs also seemed to underperform for our

classification problem. Not only was the accuracy below that of other classifiers, this particular method is much more computationally expensive than simpler and better performing methods such as kNN.