Università degli Studi di Milano

Statistical Methods for Machine Learning

Forest cover type classification using AdaBoost

Stefano De Filippis

Abstract

There are different cover types in the Roosevelt National Forest of northern Colorado, each of them is characterized by different types of trees, like spruce, aspen, some varieties of pines and other trees. The object of this work is to predict the forest cover type using the AdaBoost algorithm.

INTRODUCTION

AdaBoost is an ensemble method, these methods combine predictors able to achieve a better bias-variance tradeoff than the one obtained using the baseline predictor used in the combinations. There are many algorithms that can be seen like a combination of predictors, some of them are online learning algorithms and some are not. The predictors can be aggregated linearly or by taking a majority vote, in order to create a single classifier. In general the predictors used in the aggregation are non-parametric algorithms like tree predictors.

Considering non-online algorithms, three very popular ensemble methods are: Bagging, Random Forest and AdaBoost.

In the Bagging method, we have a training set of size m and we make a random sampling of size m T times. So we would have T training sets and each of them are fed to the algorithm A (it could be any kind of algorithm) generating T predictors. Instead, the Random Forest algorithm is similar to Bagging because it makes random sampling to the original observations but it uses only tree predictors as base algorithm. However, there is another difference: when it must make a split in the tree predictor, it considers only a subsample of the original features. In this way it adds an additional randomization which will increase the bias but decrease the variance.

In this work I use AdaBoost. This ensemble method derives from Boosting which creates classifiers of this form:

$$f = sgn(\sum_{1}^{T} w_i h_i)$$

Where w_i is a weight, h_i is the single predictor, with $i = 1, \dots, T$. In this way we have T predictors which are combined with a linear combination and the final classification is given by the sign of the result (for binary classification). The predictor h_i are very simple and they are called “decision stumps”, they usually have this form: $h_i: R^d \rightarrow \{-1, +1\}$ defined by $h_i = sgn(x_i - r)$, with $i = 1, \dots, d$ (where d is the number of features). In the case of AdaBoost this predictor is represented by a tree predictor with depth equal to 1.

Unlike Random Forest and Bagging, Boosting has got a particular characteristic, because it is able to achieve the following exponential bound on the training error:

$$l_S(f) \le \exp\left(-2T\left(\frac{1}{2} - l_{ave}\right)^2 = \exp\left(-2T\left(\frac{1}{T}\sum_{1}^{T}\right) \le e^{-2T\gamma^2}\right.\right.$$

Without requiring this independence condition:

$$\prod_{1}^{T} P(h_i(x) \neq Y)$$

The latter condition states that each classifiers make mistakes independently to the other classifiers with respect to the uniform distribution over the training set.

How AdaBoost works:

This algorithm uses the zero one loss function:

$$\mathrm{I}\{f(x_t)y_t \le 0\}$$

Where $f(x_t)$ is the prediction of the final classifier and y_t is the true label (we must remember that we are in a binary classification context). However this algorithm needs to compute also the following elements: a probability vector P, the error term ε and a vector of weights w. The probability vector P assigns a probability for each observation in the training set, so it has got a number of entries equal to the size of the training set. This vector starts assigning the same probability for each observation which is computed as $\frac{1}{m}$, where m is the size of the training set. Then for each number of rounds, this vector is updated with the following formula:

$$P_{i+1}(t) = \frac{P_i(t)e^{-w_iL_i(t)}}{E_i[e^{-w_iL_i}]}$$

Where:

$$L_i(t) = h_i(x_t)y_t$$

$$E_i[e^{-w_iL_i}] = \sum_{s=1}^{m} e^{-w_iL_i(s)}P_i(s)$$

Clearly $L_i(t) = \{-1, 1\}$, so it is equal to -1 in case of a misclassification and it is equal to 1 when the prediction is correct. The aim of the probability vector P_i is to increase the probability associated to observations which were misclassified in the previous round. Indeed, the sum of the entries of P_i is equal to one. In order to compute the error term ε_i we need the following formula:

$$\varepsilon_i = \sum_{t=1}^{m} \mathrm{I}\{L_i(t) = -1\} P_i(t)$$

So the error term is a weighted sum of errors, where the weight is given by the probability $P_i(t)$. Instead the weights w_i are computed with the method below:

$$w_i = \frac{1}{2} \ln\left(\frac{\varepsilon_i}{1 - \varepsilon_i}\right)$$

So the algorithm could be summarized like this:

```
taking a training set $S$ as input with size equal to $m$, establish the maximum number of rounds $T$
and inizialize the algorithm with $P_1(t) = \frac{1}{m}$. Then,
**for** $i = 1, \dots, T$:

    1. Feed $A$ with $S$ weighted by $P_i$ and get $h_i$ in response
    2. Compute $\varepsilon_i$ for $h_i$
    3. **if** $\varepsilon_i = \{1, 0, \frac{1}{2}\}$ then BREAK the for loop and deal with these special cases
    4. Compute $w_i = \frac{1}{2}\ln\left(\frac{\varepsilon_i}{1-\varepsilon_i}\right)$
    5. Update the probability vector $P_{i+1}(t) = \frac{P_i(t)e^{-w_i L_i(t)}}{E_i[e^{-w_i L_i}]}$, where $t$ is the single observation

**if** the loop finished with BREAK:

        **if** $\varepsilon_i = 0$:

                $f = h_i$

        **if** $\varepsilon_i = 1$:

                $f = -h_i$

        **else**:

                $f = sgn(w_1 h_1 + \cdots + w_{i-1} h_{i-1})$

**else**:

        $f = sgn(w_1 h_1 + \cdots + w_i h_i)$
```

So, except when the error term is equal to one, zero or one half, the algorithm returns a classifier which is a linear combination of predictors multiplied by the coefficients. In each round, since we feed the predictor with the probability vector P, we are focusing more on misclassified observations and at each round is possible to improve the prediction.

DATA AND METHODS

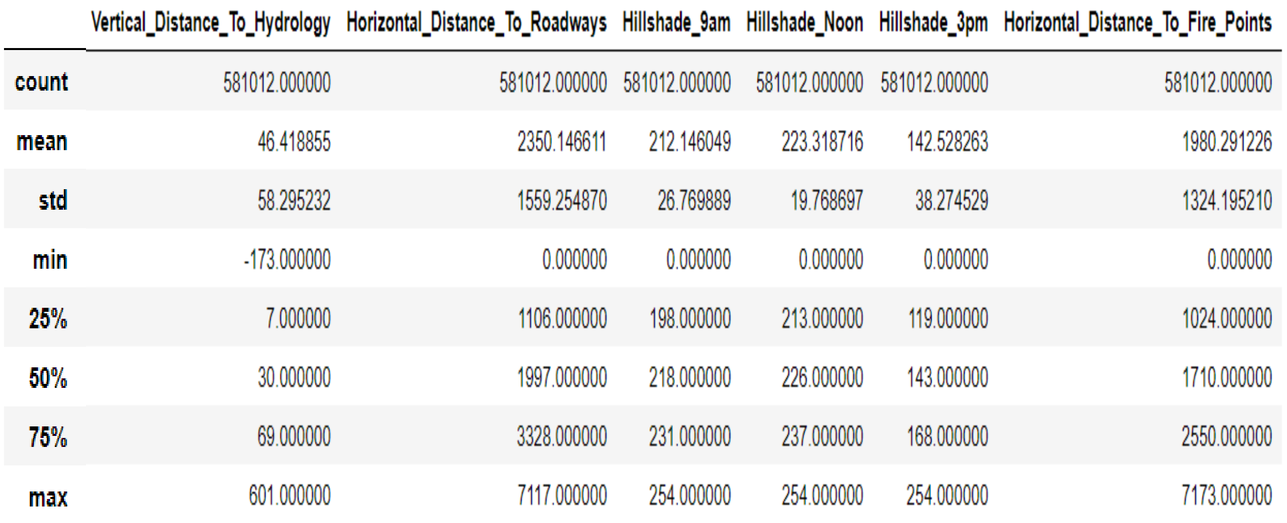
Data:

The object of this analysis is to predict the forest cover type using variables measured on trees and soils in a cell of thirty meters per thirty meters. There is a dataset easily available which includes all these variables. In particular the number of observations included in this dataset is equal to 581012 for 55 variables. These 55 variables are the followings:

- Elevation in meters
- Aspect in degrees azimuth
- Slope in degrees
- Horizontal distance to hydrology in meters (the nearest surface water)
- Vertical distance to hydrology in meters
- Horizontal distance to roadways in meters (the nearest roadway)
- Hillshade at 9am, measured with an index from 0 to 255
- Hillshade at noon, measured with an index from 0 to 255
- Hillshade at 3pm, measured with an index from 0 to 255
- Horizontal distance to fire points in meters (the nearest wildfire ignition points)
- 4 dummy variables denoting the type of wilderness: Rawah, Neota, Comanche Peak, Cache la Poudre.
- 40 dummy variables denoting the type of soil
- The variable cover type, composed by 7 classes

As stated above, the focus is on predicting the cover type, so the dependent variable is the "cover type", whereas the remaining 54 are the independent ones. In particular the cover types are characterized by a certain type of trees growing in that zone: in the class 1 there are more spruce/fir, in the class 2 there are mainly lodgepole pine, in class 3 Ponderosa pine, the class 4 is covered by cottonwood/willow, in the class 5 there are aspen, in class 6 Douglas-fir and in class 7 there are more Krummholz.

Before making any kind of analysis, it is necessary to check if there are some null values inside each column, I check this and there are zero null values for each column. At this point I start computing summary statistics like mean, standard deviation and quantiles, on all the quantitative variables.



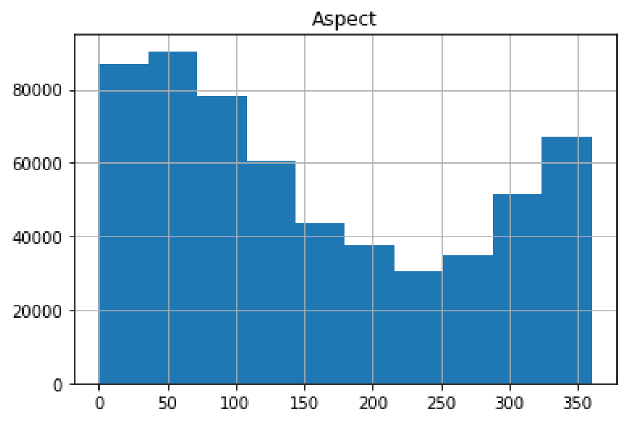
	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade_9am	Hillshade_Noon	Hillshade_3pm	Horizontal_Distance_To_Fire_Points
count	581012.000000	581012.000000	581012.000000	581012.000000	581012.000000	581012.000000
mean	46.418855	2350.146611	212.146049	223.318716	142.528263	1980.291226
std	58.295232	1559.254870	26.769889	19.768697	38.274529	1324.195210
min	-173.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	7.000000	1106.000000	198.000000	213.000000	119.000000	1024.000000
50%	30.000000	1997.000000	218.000000	226.000000	143.000000	1710.000000
75%	69.000000	3328.000000	231.000000	237.000000	168.000000	2550.000000
max	601.000000	7117.000000	254.000000	254.000000	254.000000	7173.000000

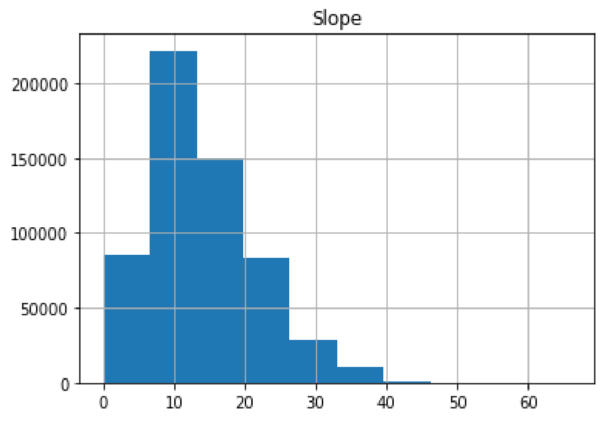
Figure 1. Extract of summary statistics

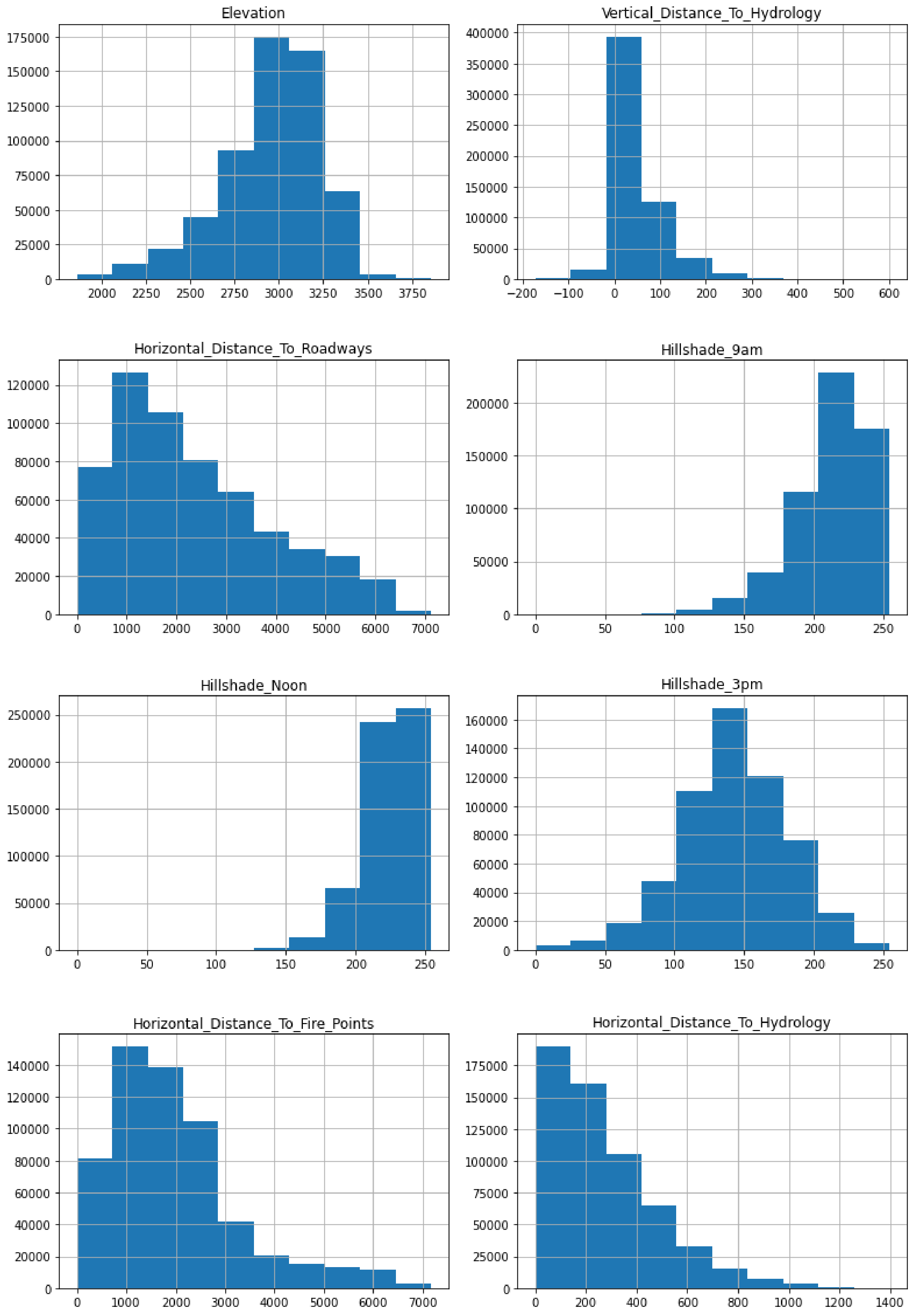
From this figure we can notice that the variables related to the hillshade have got a lower standard deviation with respect to the others, in particular the 75% of observations for the variable "Hillshade_Noon" are included in the interval 213-254.

The variables are not strongly correlated, with strongly I consider a level bigger than 0.75 in absolute value. Some of them are negatively correlated like "Elevation" and "Slope", this suggests that when the trees are in higher places the slope is getting smaller. But also the "Hillshade Noon" is negatively correlated with the "Slope". However also these correlations, despite they are ones of the highest, are not so strong and they do not overcome 0.75.

The distribution of the numerical variables are the followings:

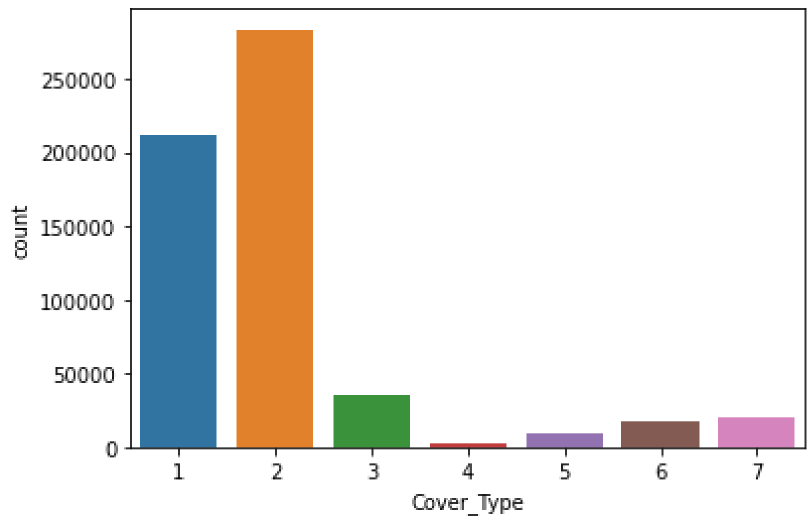





Elevation
0 25000 50000 75000 100000 125000 150000 175000
2000 2250 2500 2750 3000 3250 3500 3750
Vertical_Distance_To_Hydrology
0 50000 100000 150000 200000 250000 300000 350000 400000
−200 −100 0 100 200 300 400 500 600
Horizontal_Distance_To_Roadways
0 20000 40000 60000 80000 100000 120000
0 1000 2000 3000 4000 5000 6000 7000
Hillshade_9am
0 50000 100000 150000 200000
0 50 100 150 200 250
Hillshade_Noon
0 50000 100000 150000 200000 250000
0 50 100 150 200 250
Hillshade_3pm
0 20000 40000 60000 80000 100000 120000 140000 160000
0 50 100 150 200 250
Horizontal_Distance_To_Fire_Points
0 20000 40000 60000 80000 100000 120000 140000
0 1000 2000 3000 4000 5000 6000 7000
Horizontal_Distance_To_Hydrology
0 25000 50000 75000 100000 125000 150000 175000
0 200 400 600 800 1000 1200 1400

Among these variables the only one that has got a shape similar to a normal distribution is the one related to hillshade at 3pm.

Instead, the dependent variable related to the cover type is the one shown below:



It is possible to notice that the distribution of classes is not balanced: the first two types are the most observable in the dataset, but the other classes instead are very less represented, in particular the fourth class counts 2747 observations against 211840 for the second forest cover type.

Methods:

I proceed splitting the dataset into a training and a test set, in the first there are 66% circa of the observations and in the latter around 33%. I build the AdaBoost algorithm from scratch using the formulas written in the introduction. However, the problem analyzed is a multiclass classification, so it is necessary to make a one vs all classification. Basically, I create an AdaBoost classifier for each class for a total number of seven. Each classifier creates a predictor which predicts if an observation belongs to that class or not, in the first case the prediction will be equal to 1 and in the latter will be equal to -1. Of course, the final prediction will depend on the sign of the linear combination of predictor multiplied by coefficients (except in the special case written in the pseudo algorithm in the introduction) as this formula remarks:

$$f = sgn(w_1 h_1 + \cdots + w_i h_i)$$

If the sign is negative the prediction will be -1 otherwise it will be equal to 1.

However, all these binary classifiers must be combined in order to make a multiclass classification, the final prediction could be explained with this formula:

$$arg \max_k \frac{1}{T^k} \sum_{i=1}^{T^k} w_i^k \, h_i^k(x)$$

Where:

- k is one of the seven classes
- T^k is the number of decision stumps used in the AdaBoost for the class k
- w_i^k is the coefficient estimated in the round i for the class k
- $h_i^k(x)$ is the binary prediction made in the round i for the class k

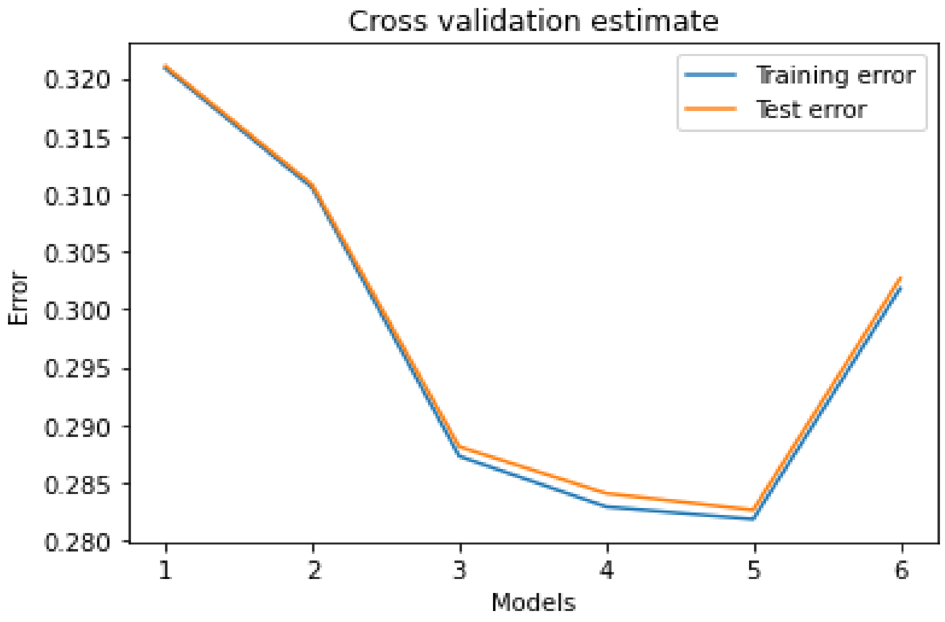
So I decide to use a number of decision stumps different for each class, because, as shown before, the distribution of the dependent variable is not balanced, so probably some classes need a bigger number of rounds in other to adjust the prediction. At the end of the rounds, it takes the maximum value of the linear combinations and the label predicted will correspond to the one attached to the maximum value.

The training set contains 389278 observations and it is used to make the cross validation. This method is very useful to estimate the risk of a predictor, because this one is expressed as the expectation of the loss given the distribution D. So if we know D we will be able to estimate the risk, but at the same time we would not need to solve a learning problem since the distribution of the dependent variable is known. So, considering that D is unknown, the cross validation method could help in estimating the risk. It splits the dataset into a number of folds and it runs the algorithm in all of them except one which is used as test set. The entire procedure is repeated so that each fold in turn is a test set. Then the risk is computed taking the average of the test errors. I decide to use a number of folds equal to five, so the entire procedure is repeated five times, where each time the training set is composed by 311422 observations and the test by 77856 observations.

I use this method also to choose the number of decision stumps for each class, I start with the variables shown before without making any kind of manipulation. So I include all the variables also the ones with apparently a small variance, I justify this choice because excluding variables means losing information.
The number of decision stumps chosen are the followings:

1. [10, 8, 20, 70, 60, 50, 40]
2. [50, 70, 30, 15, 20, 25, 30]
3. [300, 400, 160, 30, 50, 60, 70]
4. [400, 400, 400, 400, 400, 400, 400]
5. [500, 600, 400, 100, 150, 170, 180]
6. [150, 100, 200, 600, 500, 400, 300]

These are six models, for each of them it is specified the number of stumps, the order defines also the class, so in the first position there is the number of stumps for the first class, in the second position for the second class and so on up to seven. I start with a small number of stumps and then I increase them, but in different ways: in some model I put more emphasis on the most common classes inside the dataset (the first, second and the third class) , instead in other ones I use more stumps on the less common classes (the fourth, fifth, sixth and the seventh) because they are more difficult to predict and increasing the number of stumps maybe could help since in each round AdaBoost is trying to correct observations which have been misclassified in the previous rounds. This plot shows the results:



The first thing that can be noticed is that the test and training error decreases when the number of stumps is bigger, because the models from left to right have an increasing number of stumps. However, when I put more emphasis on less represented classes the error is bigger than the opposite case. The first and the second model are similar in terms of total stumps used, but in the first model the first two classes have few decision stumps and the test error is about 0.32108, instead when the number of stumps for those classes is increased, the test error is 0.31147. In the model 3 the number of stumps is bigger for all the classes than the previous two models, but the first three classes have got a larger number of stumps in relation to the other cover types. Here the test error is again decreased, reaching the value of 0,28813. The model 4 has got a number of stumps equal to 400 for each class and also here the test error is decreased. This confirms that increasing the number of stumps contributes to decreasing the test error. However, for the model 5 and 6 the situation is different: the first puts more emphasis on the first three classes, instead the latter makes the opposite. The model 5 has got the lowest test error and it is equal to 0,28266. The model 6 has got a smaller accuracy than the 5, 4, 3, this suggests that probably is better to

increase the number of stumps for the first classes instead to use a large amount of stumps for the last classes.
Another thing that could be noticed, is that the training and test errors for these models are very similar, so none of these models lead to a situation of overfitting where the training error is small and the test error is large.

Among these options, the final choice is the model 5, so I train it again using all the training set observations and the accuracy for the training set is 71,884%, instead for the test set is 71,878%.
In order to analyze better the accuracy of the model, it is useful to compute a confusion matrix, the result for the test set is the following:

		PREDICTED LABELS							
TRUE LABELS		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Total**
	1	49662	19560	24	0	0	0	661	**69907**
	2	17450	74824	1095	4	0	84	32	**93489**
	3	0	1727	9730	118	0	224	0	**11799**
	4	0	7	376	478	0	46	0	**907**
	5	2	3058	73	0	0	0	0	**3133**
	6	0	1953	3355	27	0	396	0	**5731**
	7	3996	47	0	0	0	0	2725	**6768**
	Total	**71108**	**101176**	**14653**	**627**	**0**	**750**	**3418**	**191734**

On the main diagonal there are the observations classified correctly, thanks to this table it is possible to compute the accuracy for each class. For the first class the number of observations in the test set is 69907, the correctly classified are 49662, but 19560 are classified with the class 2, instead 24 are classified as 3 and 661 observations are classified as 7. The overall accuracy for this class is equal to 71,04%. The same considerations could be done for the other classes: the accuracy for the class 2 is equal to 80,04%, for class 3 is qual to 82,46%, for class 4 is equal to 52,7%, for class 5 is equal to 0%, for class 6 is equal to 6,91% and for class 7 is equal to 40,26%. So, this model performs well for the first three classes which are also the most common in the dataset, instead for the remaining ones the results are not satisfying. For the class 5 the AdaBoost classification is completely wrong and for the class 6 the situation is slightly better.

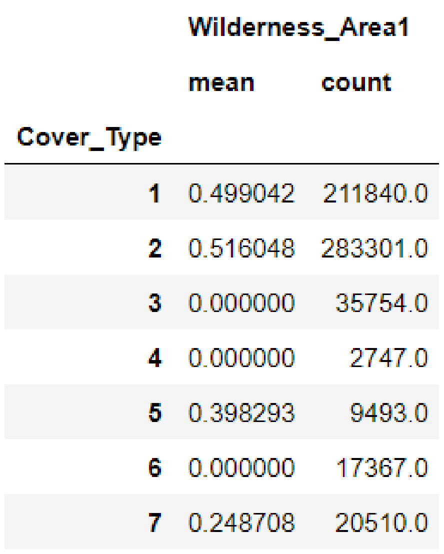
However, the model 4 has got an accuracy slightly smaller than the model 5 and the number of stumps for each class is equal to 400, so it could be useful also to analyze its performance on the whole training set. In this case the accuracy on the training set is equal to 71,62%, whereas in the test set is equal to 71,64%, so also this model does not overfit. Then I compute the confusion matrix. The result is the following.

		PREDICTED LABELS							
TRUE LABELS		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Total**
	1	49554	19709	24	0	0	2	618	**69907**
	2	17675	74469	921	6	0	349	69	**93489**
	3	0	1696	9271	123	0	709	0	**11799**
	4	0	3	407	442	0	55	0	**907**
	5	0	3052	81	0	0	0	0	**3133**
	6	0	1669	2911	33	0	1118	0	**5731**
	7	4226	39	0	0	0	0	2503	**6768**
	Total	**71455**	**100637**	**13615**	**604**	**0**	**2233**	**3190**	**191734**

In this case the accuracy of the first four classes and the seventh one is decreased, but not so much. In particular, the first class shows an accuracy about 70,88% against 71,04% of the previous model. For the second class the situation does not change a lot, the accuracy here is equal to 79,66% whereas in model 5 is 80,04%. In the third class the decrease of the accuracy is a bit bigger, here it is equal to 78,57% instead with the previous model is about 82,46%. But the fourth class is the one which suffered a more decrease in the accuracy, here it is about 48,73% instead in the model 5 is equal to 52,7%. In the class seven the accuracy shown by this model is equal to 36,98% whereas in model 5 is 40,26%. However, the performance for the class 6 is increased a lot, it passes from 6,91% to 19,51%, but this value is still very low. Also in this case the performance for the class 5 is completely wrong with an accuracy of 0%.

Between these two models I would prefer the second one, because it is true that the overall performance is slightly worse than the model 5, but this difference is very small and about 0,238%. In addition, it performs better for the class 6, even if the accuracy remains poor, the difference here between these models is bigger. Consecutively I would be willing to lose some accuracy in the other classes in order to have a better performance in the class 6, instead of having better accuracy in the other classes but an almost completely wrong prediction for that class.

However, the class 5 remains very difficult to be predicted, in particular, watching to the two confusion matrices, it is possible to notice that the majority of observations belonging to this class are mistakenly classified with the class 2. So probably there are some similarities between the variable measured in the soil where these two cover types are present. For example, the second class is very common in the wilderness area of Rawah (wilderness area 1) and Comanche Peak (wilderness area 3), as well as the class 5. So, this could be one of the possible reasons of this misclassification, because the number of observations belonging to the class 2 is far bigger than the one related to class 5. This thing could be also one of the reasons why other classes, like the first and the seventh are classified as 2. The image below shows the percentage and the count of observations belonging to the wilderness area 1 (the Rawah area) for each class. It is evident that the majority of observations are from class 2, but there are also observations from class 1, 5, and 7. However, since the class 2 has got more observations than the others, this could influence the prediction, because a tree predictor predicts the most common class observed in the examples routed to a leaf.



	Wilderness_Area1	
	mean	**count**
Cover_Type		
1	0.499042	211840.0
2	0.516048	283301.0
3	0.000000	35754.0
4	0.000000	2747.0
5	0.398293	9493.0
6	0.000000	17367.0
7	0.248708	20510.0

In some learning problems it is useful to manipulate variables in order to give them a shape similar to a normal distribution. There are different ways to make this thing, the main methods are: taking the logarithm or the square root. In this dataset there are variables whose values sometimes are equal to zero or negative, so it is not possible to use these two methods. But a cube root could help in obtaining a shape more similar to a normal. However, tree based algorithms do not benefit from a monotone transformation of variables. So this is a method not considered in this work.

CONCLUSION

The object of this analysis is to predict the forest cover type using 54 variables measured on the soil in a cell of 30 meters per 30 meters. The classes that must be predicted are seven and they represent different types of trees: spur/fir, Lodgepole pine, Ponderosa pine, cottonwood/willow, aspen, Douglas-fir, Krummholz. In this analysis is used an ensemble method, these methods combine predictor in order to achieve a better variance-bias trade off. The AdaBoost is the one used in this work. In this algorithm the base predictor is a decision tree with depth equal to one called “decision stump”. The AdaBoost works computing errors, vector of probability and weights. In each round of this algorithm the three elements mentioned before are updated and the vector of probability has got an important role: it puts more emphasis on observations misclassified in the previous round, in this way the prediction could be corrected in the future rounds. But since the classes are seven, I compute seven different AdaBoost classifiers. In order to choose the best number of stumps, I evaluate different models using the cross validation, the number of stumps in each model are the following:

- Model 1 – [10, 8, 20, 70, 60, 50, 40]
- Model 2 – [50, 70, 30, 15, 20, 25, 30]
- Model 3 – [300, 400, 160, 30, 50, 60, 70]
- Model 4 – [400, 400, 400, 400, 400, 400, 400]
- Model 5 – [500, 600, 400, 100, 150, 170, 180]
- Model 6 – [150, 100, 200, 600, 500, 400, 300]

The results of the cross validation show that increasing the number stumps makes the prediction more accurate, but in different ways. The model 6 has got in general more stumps than model 5, but its accuracy is equal to 69,73% instead in the model 5 the accuracy is better and equal to 71,73%. This result seems suggesting that putting more emphasis on the most common classes leads to better models instead of making the opposite. Between all these models the fifth one reaches the best accuracy, so I decide to train it with all the observations in the training set and it reaches an accuracy of 71,878% in the test set. However, the confusion matrix shows that this model performs very bad in predicting the class 5 and 6. In the first one the accuracy is zero, instead in the latter it is equal to 6,91%. So I decide to analyze also the model 4 which has got a cross validation error similar to the model 5 but slightly bigger. The overall performance of this model trained on all the training observations shows results slightly worse for the class 1,2,3,4 and 7, but for the class 6 the accuracy is around 19,51% which is better than the accuracy reached with the model 5. Also in this model the accuracy for the class 5 is equal to zero. Taking care of the difference between these two models, I would choose the model 4 because it could guarantee a better performance for the sixth class, even if also in this case the accuracy is poor. The observations belonging to class 5 are almost always classified as 2, one possible reason is that the latter is the most common class in the dataset and this could affect the prediction.

REFERENCES

- Course materials of the course Statistical methods for Machine Learning
- UCI Machine Learning Repository: Covertype Data Set
- https://www.kaggle.com/datasets/uciml/forest-cover-type-dataset