Predicting forset cover type from cartographic variables only

Cristian Di Pietrantonio

February 14, 2017

Abstract

This report has the objective to show how different classifiers perform on the forest cover type problem. In particular, Naive Bayes, Neural Network, Decision Tree and Random Forest are subject to accuracy testing, cross validation and learning curve analysis to establish which one is best suited to model the phenomenon.

Contents

1 Introduction

The objective of this work is to develop and compare different models of a phenomenon using machine learning techniques. In particular, the interest lays in predicting the forest cover type of a region given certain characteristics of the soil and the environment.

In order to do so, a large dataset of 581012 instances is provided, where each instance is an observation of a 30×30 meter cell characterized by 54 attributes and associated with the correct class of forest cover type. The 54 columns of data are divided as follows: 10 quantitative variables, 4 binary wilderness areas and 40 binary soil type variables.

It is thus a classification problem. To find the best model different machine learning algorithms are evaluated, each one with carefully selected parameters; next, a comparison of their predictive capacity (accuracy) and some other metrics takes place to find out which type of classifier best fit the problem to be solved. In particular, the following supervised learning algorithms are evaluated: *Naive Bayes*, *Neural Networks*, *Decision Tree*, and *Random Forest*. The choice has not been casual: each one of these classifiers works in a different way from the others, so that the probability of finding the best approach is maximized.

2 Experimental settings and planning

In this section experimental settings, that are machines and tools used, are described. Furthermore, the experiments' structure will be laid out, indicating which set of tests classifiers are subject to in order to be evaluated.

2.1 Hardware and software

Experiments will be performed on a laptop having 12Gb og RAM DDR3, an Inter Core i7 processor with a maximum clock frequency of 3.4ghz and a Nvidia GeForce 820M GPU.

On the software side, Python has been selected as programming language for its ease of use and the vast number of good machine learning libraries. In particular, the module implementing machine learning algorithms that has been chosen is scikit-learn; it is the most used machine learning Python library in the scientific community, so it is well tested and optimized. Following the guidelines of Python, only few lines of code are needed to run a classifier in scikit-learn. The typical workflow is the following:

- Initialize a new classifier object, for example a Multilayer Perceptron, passing the needed parameters to the constructor. Parameters are specific for each classifier.

```
clf = MLPClassifier(hidden_layer_sizes=(25))
```

- Fit the classifier using training data:

```
clf.fit(feature_vectors, labels)
```

- Finally, give the classifier instance unseen instances and get a predicted label for them:

```
predictions = clf.predict(unseen_instances)
```

Every classifier implements the *fit* and *predict* methods. Furthermore, scikit-learn has a lot of functions to manipulate data and perform testing and cross-validation.

Even though scikit-learn implements a lot of machine learning functions, a series of helper functions have been implemented in first person. For example, a class *Dataset* has been created to deal with the dataset and it is capable of: computing the information gain; splitting the dataset randomly in K subset of the same size; creating a subsample of the original dataset in such a way to keep the same class distribution (this is useful when testing different parameters without spending too much computation time on it).

2.2 Experiment structure

The final objective is to compare different classifiers, so it is mandatory that they all have to be subject to the same set of tests and evaluation criteria. The following phases are established:

1. *Preprocessing*: the dataset will be subject to a series of manipulations to make classifiers find an optimal fit more easily and in less time;
2. For each classifier C:
   (a) *Best parameters search*: given C, different values for its parameters will be tested in order to find a satisfying combination;
   (b) *Accuracy Testing*: the capabilities of classifier C are first tested using a dataset split. More precisely, it is split in *training* and *test* set, the latter having 37% of the instances in the original dataset;
   (c) *K-Fold Cross Validation*: the accuracy of classifier C is assessed using K fold cross validation, with $K = 30$. In this way we can estimate the mean accuracy and its 95% confidence interval;
   (d) *Learning curve*: as last test, accuracy is observed as a function of the dataset size. We can then recognize whether a classifier's performance depends heavily on the data quantity.
3. *Classifiers' performances comparison*: having gathered the data, a comparison takes place to establish which one can best fit the prediction task.

3 Preprocessing

The first step involves data analysis and preprocessing. It is important to assess the quality and quantity of information contained in the dataset; these traits can influence and explain particular behaviors of a classifier. Moreover, data can be in a form not yet suitable for training a classifier, and must be manipulated to guarantee the best performances in terms of quality of prediction and time.

3.1 Entropy and Class Distribution

Entropy is a measure of uncertainty of a random variable. In this case the random variable is the class attribute in each example in the dataset. So it gives a measure of how much instances in this dataset are predictable. Entropy is computed using the following formula:

$$Entropy = \sum_{c \in C} \frac{n_c}{n} \log_2 \left(\frac{n_c}{n} \right), \tag{1}$$

Where C is the set of labels (or classes) and n_c is the number of instances belonging to class c. In our case entropy is equal to 1.73 (the maximum value is 2.8) or, equivalently, 61%. This means that instances are not equally distributed amongst classes. Considering the class attribute as a random variable, its distribution is shown in Table 1. To make more evident the disparity in class distribution, a bar chart with the probabilities is shown in Figure 1.

Class	**Probability**
1	0.36461
2	0.48760
3	0.06154
4	0.00473
5	0.01634
6	0.02989
7	0.03530

Table 1: class distribution.

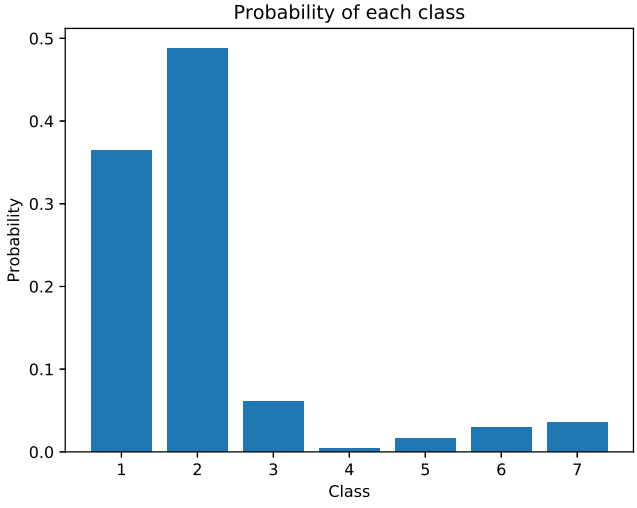

Figure 1: class distribution in a bar chart.

Classes 1 and 2 hold most of the instances, followed by class 3. Class 4 has a very low frequency, so one can imagine that will be harder to learn to recognized

it, in contrast with the first two. This dataset gives the possibility of studying classifiers' performances in non optimal conditions.

3.2 Information Gain of Attributes

To have an idea on how much each attribute contributes in determining the label of an example, *information gain* of attribute j is computed using the following formula:

$$InformationGain = Entropy - \sum_{v \in V_j} \frac{n_v}{n} Entropy(S_v), \tag{2}$$

where S_v is the set of instances that have value v for feature j. Information gain is computed over the dataset, for each attribute, and the results are shown in Table 2.

Attribute ID	Absolute value	Percentage value
0	0.6662	38.32
13	0.2108	12.12
5	0.1513	8.70
9	0.1225	7.05
10	0.0962	5.53
23	0.0923	5.31
2	0.0526	3.02
6	0.0463	2.66
51	0.0440	2.53
42	0.0428	2.46
52	0.0419	2.41
17	0.0405	2.33
3	0.0367	2.11
35	0.0347	2.00
25	0.0344	1.98
15	0.0338	1.95
8	0.0325	1.87
7	0.0324	1.86
1	0.0316	1.82
36	0.0304	1.75
19	0.0288	1.66
4	0.0287	1.65
53	0.0255	1.47
16	0.0210	1.21
11	0.0192	1.10
14	0.0179	1.03
43	0.0167	0.96

26	0.0141	0.81
45	0.0128	0.73
46	0.0104	0.60
24	0.0103	0.59
12	0.0103	0.59
30	0.0095	0.55
18	0.0094	0.54
44	0.0076	0.43
37	0.0072	0.41
48	0.0069	0.39
27	0.0043	0.24
31	0.0027	0.16
33	0.0026	0.15
50	0.0025	0.14
39	0.0025	0.14
32	0.0021	0.12
34	0.0018	0.10
47	0.0016	0.09
41	0.0012	0.07
22	0.0010	0.06
29	0.0009	0.05
40	0.0004	0.03
49	0.0004	0.02
38	0.0003	0.01
20	0.0002	0.01
21	0.0001	0.01
28	0.0000	0.00

Table 2: Attributes ranked by Information Gain.

The first two attributes, namely *Elevation* and a binary column of the abstract attribute *Wilderness Area*. At the opposite side, we have the 28^{th} attribute that has no influence on class distribution. In order to make computations faster, attributes with information gain less than 0.1% are removed (9 attributes).

Next, quantitative attributes are normalized to speed up computation (for example, a gradient descent algorithm could take a significant amount of time to converge if the range of values is large) and to give each attribute equal weight in distance-based machine learning algorithms (k-means, for example).

Here ends the preprocessing work done on the dataset. All other attributes take binary values so there is nothing that can be done to reach noticeable better performances.

4 Naive Bayes

The first classifier to be tested is Naive Bayes. A Naive Bayes classifier assumes the independence of the components in the feature vector and compute the probability $P(y_i|\mathbf{x})$ of label y_i given the observed instance $\mathbf{x}$ using Bayes' theorem:

$$P(y_i|\mathbf{x}) = \frac{P(y_i)P(\mathbf{x}|y_i)}{P(\mathbf{x})} = \frac{P(y_i)\prod_{j=1}^{n} P(X_j = v_{jk}|Y = y_i)}{P(\mathbf{x})}, \tag{3}$$

with $i = 1, \ldots, m$, where n is the number of features, m the number of examples. The theorem allows us to compute needed probabilities from the dataset. For this classifier, two experiments will be performed: in the first it is assumed that the features underlying distribution is Gaussian, in the second a multinomial one.

As for the parameters tuning, the only ones that can be set are the *priors*. It has been decided to apply Laplace Smoothing, which is the default option and a reasonable one.

4.1 Gaussian Naive Bayes

Since some features take on real values one can think that their underlying distribution is Gaussian and mean and variance can be estimated for feature X_j given that the example belongs to class y_i. Then, the probability of each feature value is computed using the following formula:

$$P(X_j = v_{jk}|Y = y_i) = \frac{1}{\sigma_{ji}\sqrt{2\pi}} exp\left(\frac{-(X_j - \mu_{ji})^2}{2\sigma_{ji}^2}\right) \tag{4}$$

4.1.1 Accuracy Testing

A first test is run to assess classifier's capacity using a dataset split, training the model with 63% of the data and testing it with the remaining instances. The resulting accuracy is equal to 17.40% or, equivalently, the error is equal to 82.6%. It is clear that the current classifier performs very poorly with this dataset. Table 3 shows detailed statistics for each class; Table 4 shows the confusion matrix. As can be seen we have high precision on the classes (1 and 2) having the major number of instances, but recall capacity is really low. This suggests that lack of equidistribution have played an important role in causing poor performances.

Class Index	Precision	Recall	F-Measure
1	69.830	14.093	23.453
2	90.310	9.601	17.356
3	27.364	39.720	32.404
4	7.557	100.000	14.051
5	2.431	71.957	4.702
6	18.282	6.362	9.439
7	14.648	93.715	25.335

Table 3: precision and recall for each class.

1	2	3	4	5	6	7	← classified as
14.09	1.37	1.31	0.00	46.49	0.95	35.80	1
4.49	9.60	9.40	1.06	62.17	1.01	12.27	2
0.00	0.00	39.72	59.67	0.45	0.17	0.00	3
0.00	0.00	0.00	100.00	0.00	0.00	0.00	4
0.17	0.00	26.43	0.00	71.96	0.82	0.62	5
0.39	0.00	32.02	55.87	5.11	6.36	0.25	6
0.79	0.00	1.00	0.00	4.50	0.00	93.72	7

Table 4: Confusion matrix for Gaussian Naive Bayes.

4.1.2 Cross Validation

The next step is to apply K-fold cross validation, with $K = 30$, to assess the classifier's performance with reasonably confidence in what will be discovered. Accuracy in each trial (fold) is computed and the resulting mean accuracy is 17.468%, with a 95% confidence interval of ± 0.110.

As can be seen in Figure 2, in each fold the classifier has a very low accuracy. This explains why the confidence interval is small around the mean. Cross validation confirmed what was saw before: Naive Bayes, with the Gaussian assumption, is unable to predict the cover type. Furthermore, precision, recall and f-measure of each class, shown in Table 5 and Figure 3, show a great unbalance between precision and recall. Classes 3 has the highest *harmonic mean* (that is, F-measure), while on class 4 the classifier shows full recall capability (this may be due to the fact that there are really few class 4 instances). Overall, F-measure values are really low, confirming that this classifier would be a poor choice.

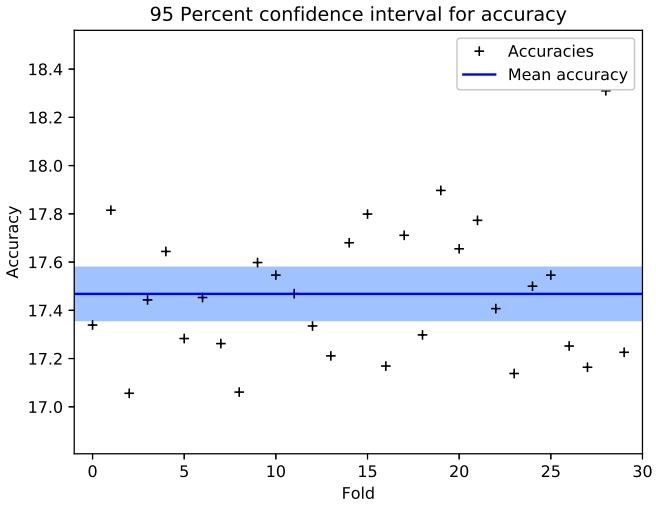

Figure 2: Mean accuracy and confidence interval for Gaussian Naive Bayes.

Class Index	**Precision**	**Recall**	**F-Measure**
1	69.276	14.120	23.456
2	90.805	9.628	17.409
3	27.295	39.456	32.260
4	7.430	100.000	13.819
5	2.512	74.007	4.859
6	14.758	6.656	9.161
7	14.634	93.974	25.320

Table 5: mean precision, recall and F-measure using cross validation.

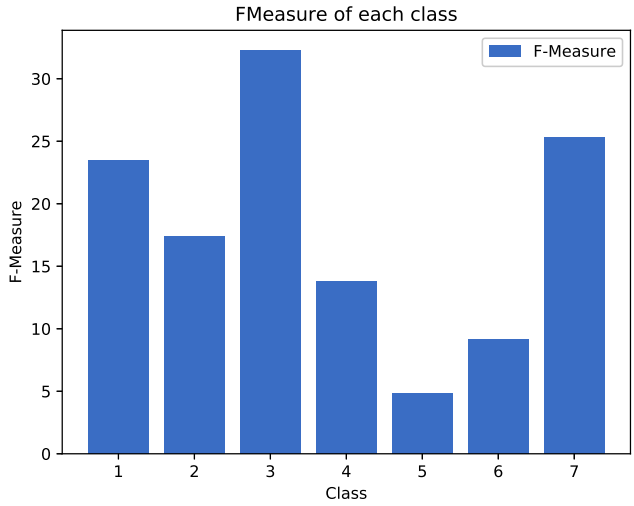

Figure 3: F-measures resulting from cross validation.

4.1.3 Learning curve

Dataset Size	Accuracy
58102	12.350
116203	12.966
174304	12.639
232405	17.330
290506	17.266
348607	17.525
406708	17.525
464809	17.537
522910	17.536
581011	17.473

Table 6: Dataset sizes and relative accuracy.

As final test, the learning curve of the classifier is studied. The main dataset is divided in ten parts and another ten datasets, of growing size, are created. Can the classifier improve its performance with a lager set of data? To answer this question, Naive Bayes is tested on the newly created datasets, and every accuracy value is annotated in Table 6. The minimum value is 12.3, associated with the smallest dataset; the maximum value is 17.537, obtained using one of the largest dataset. It can be said that there has been an improvement, even though a small one considering the datasets' size. Looking at Figure 4 it is clear that size $200k$ acts as the threshold for improvement. The first three points are stable around 12.5; then there is a jump to 17.3 and from that point on the

learning curve becomes flat at that value. It follows that most likely a greater dataset cannot improve the classifier's performance.

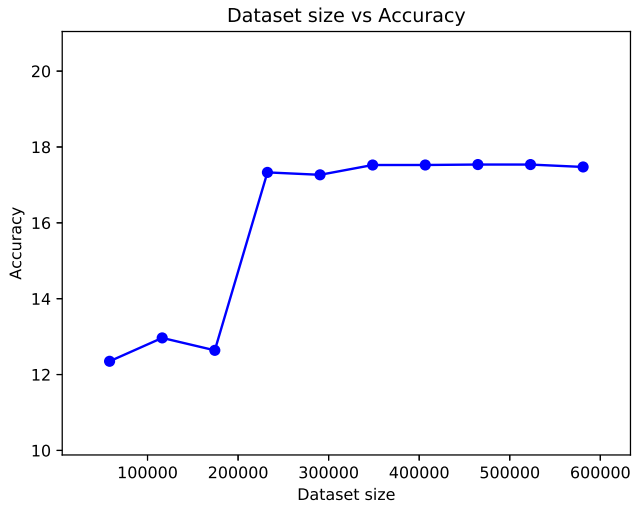

Figure 4: Learning curve of Gaussian Naive Bayes.

4.2 Multinomial Naive Bayes

Now the distribution assumption on features' values is changed to a multinomial one. Feature vectors represent the frequencies with which features have been observed and are modeled by a multinomial distribution.

4.2.1 Accuracy Testing

As before, the dataset is split in test set and training set; a multinomial Naive Bayes classifier instance is run. The accuracy registered is equal to 64.013% and, even though it is not a good enough value, it shows a huge improvement in performance with respect to the Gaussian Naive Bayes classifier. Detailed statistics are listed in Table 7. Table 8 shows the confusion matrix. The first thing to be noticed is that class 5 is not recognized at all; in the case of the Gaussian distribution, it had the lowest F-measure, 4.8, and it is the second lowest frequent class in the dataset. Most of class 5 instances are classified as belonging to class 2, as can be seen in the confusion matrix. Class 4 and 6 are still hard to recognize, in fact a lot of their instances are misclassified with label 3. On the other hand, all the other classes present higher values for both precision and recall, and these values are balanced.

Class Index	Precision	Recall	F-Measure
1	65.315	46.433	54.278
2	64.730	80.886	71.912
3	55.646	92.842	69.585
4	65.730	11.448	19.500
5	0.000	0.000	0.000
6	43.750	2.615	4.935
7	68.483	49.605	57.535

Table 7: precision, recall and f-measure of classes using multinomial Naive Bayes.

1	2	3	4	5	6	7	← classified as
46.43	51.38	0.11	0.00	0.00	0.00	2.08	1
15.32	80.89	3.66	0.00	0.00	0.00	0.13	2
0.00	5.96	92.84	0.42	0.00	0.78	0.00	3
0.00	0.00	77.89	11.45	0.00	10.67	0.00	4
8.14	82.18	9.67	0.00	0.00	0.00	0.00	5
0.90	23.99	72.43	0.06	0.00	2.62	0.00	6
36.33	13.64	0.43	0.00	0.00	0.00	49.61	7

Table 8: confusion matrix for multinomial Naive Bayes.

4.2.2 Cross validation

It is now the moment to apply cross validation. The mean accuracy obtained is 64.165%, with a 95% confidence interval of ±0.103, shown in Figure 5. This result is in accordance with what has been found in the previous test. The small confidence interval indicates that the sample mean value is indeed representative of the true accuracy of the classifier. As has already been said, 64% is still a non satisfactory value, but represents a great improvement to the Gaussian classifier's one. Detailed statistics for each class are shown in Table 9. Even in cross validation class 5 is still not recognized, while other classes present a precision percentage around 60%. Recall values are more sparse, going from 2.4% to 92.9% (excluding class 5). F-Measure values, as can be seen also in Figure 6, are fairly high for classes 1, 2, 3 and 7. It is interesting to note that class 7 has a relatively high f-measure, even though its instances are not frequent in the dataset, with respect to class 1 and 2 instances; yet, they present approximately the same F-measure. It may be that exist feature values particular to that class of instances.

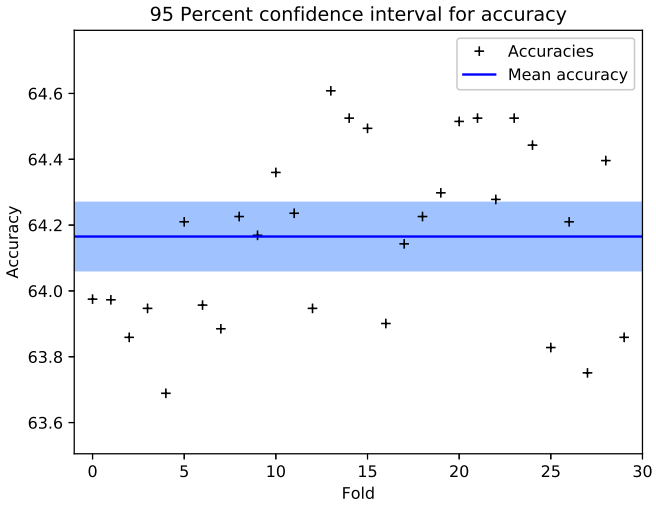

Figure 5: Mean accuracy and confidence interval for Gaussian Naive Bayes.

Class Index	**Precision**	**Recall**	**F-Measure**
1	66.014	46.460	54.536
2	64.665	80.939	71.892
3	55.770	92.988	69.716
4	67.020	12.332	20.686
5	0.000	0.000	0.000
6	46.617	2.483	4.710
7	68.271	53.977	60.265

Table 9: mean precision, recall, F-measure for multinomial Naive Bayes in cross validation.

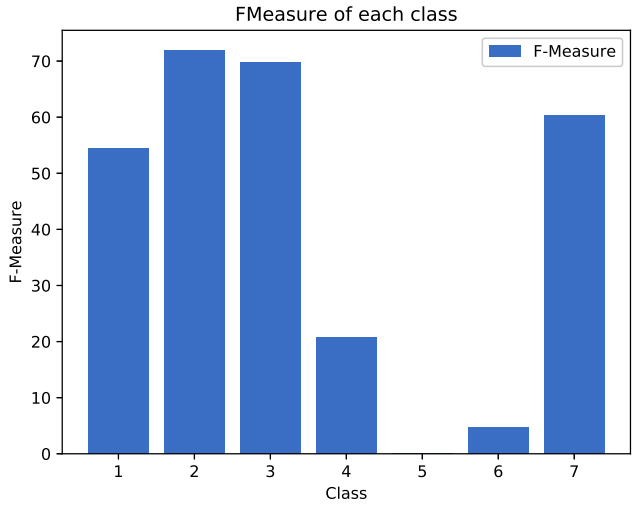

Figure 6: F-measure of classes resulting from cross validation.

4.2.3 Learning curve

The classifier's accuracy has been evaluated over different dataset sizes and results are shown in Table 10; the learning curve for multinomial Naive Bayes classifier is shown in Figure 7. In this case accuracy seems to be independent from the dataset size, since it is stable at 64%.

Dataset Size	**Accuracy**
58102	64.536
116203	64.041
174304	64.114
232405	64.504
290506	64.086
348607	64.055
406708	64.075
464809	64.168
522910	64.185
581011	64.266

Table 10: Dataset sizes and relative accuracy for multinomial Naive Bayes.

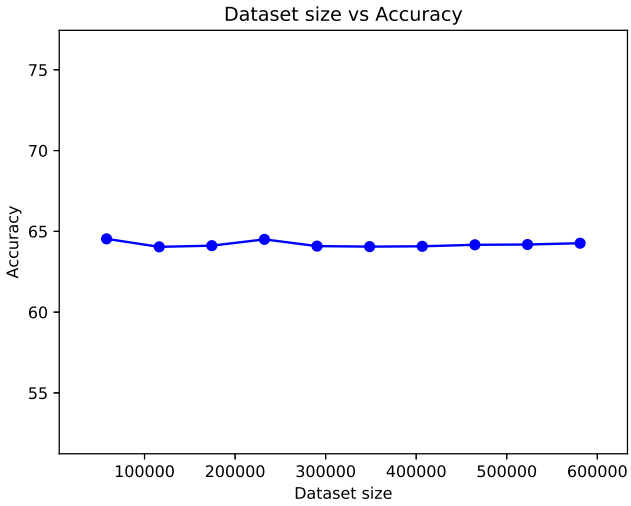

Figure 7: Learning curve for multinomial Naive Bayes.

5 Neural Networks

Neural networks are a computational approach that loosely model the way biological brain solves problems. The model is composed by several units, called *perceptrons*, organized in multiple *layers*, where each unit in layer i is *activated* when receiving a *signal* from all the units in layer $i-1$. The first layer represent the input vector, the last layer encodes the output. There is a weight on each edge connecting two perceptrons, which proportionally amplify the signal on that edge. The problem is to find a set of weights that make a neural network predict the right output for a given input vector.

5.1 Parameters selection

The first decision to take when using neural networks to solve a problem regards its architecture. In particular the number of *hidden layers* and the number of perceptrons in each layer must be defined. Since we will use the backpropagation algorithm, the number of hidden layers suggested in literature is one. Since the number of input units and output units is fixed, it remains to decide how many hidden units are needed. What has been done is defining a search space and try every value in it, using a *subsample* of the original dataset (the reason is to avoid too long computation time just for parameter tuning). Another parameter to be chosen is the maximum number of *epochs* that a neural network goes through during the training phase: 300 is considered a reasonable value given the size of the dataset.

The search space defined is

$$\mathcal{S} = \{10, 15, 18, 20, 23, 25, 27, 30, 35, 40, 45\}$$

that is, numbers between n and $\sqrt{n}$, where n is the number of features. As for the subsample, an apposite function was manually implemented to pick a set

of instances of a given size from the original dataset such that the distribution of class values does not change. In this case, the subsample contain 5% of the original number of instances (which is still a great number of instances, 29000).

For each value in $\mathcal{S}$ we build a neural network with that number of hidden units and test it with 5-fold cross validation. The number of hidden units associated with the highest score is chosen.

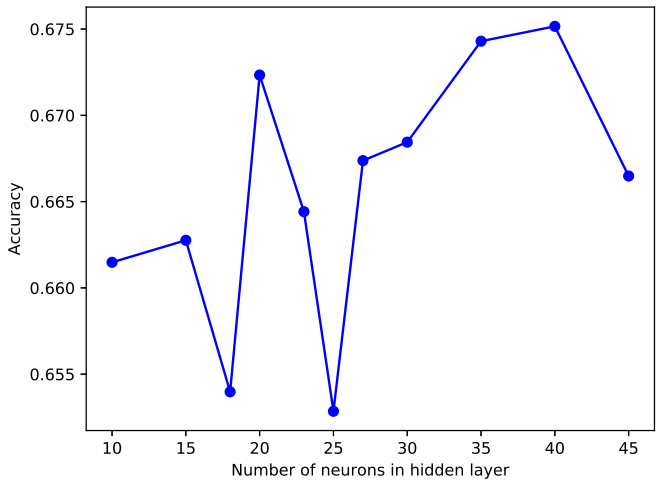

Figure 8: Scores obtained variating the number of hidden units in a neural network.

As can be seen in Figure 8, in general the more hidden units more is the score obtained. The highest score is associated with 40 hidden units, so that values is chosen by the automated script that performs the various tests. It is to be noticed that the improvement in accuracy is not noticeable so a smaller number of hidden units would have been fine (it is also true that these results are valid on the *subsample*; maybe a major number of hidden units is needed to handle the original dataset well).

5.2 Accuracy Testing

Now that parameters has been chosen, in order to have a general idea of its capabilities, the neural network is trained using 63% of the instances in the dataset and tested using the remaining 37%. The accuracy of the neural network is 79.908%, which is a decent value, considering the non optimal distribution of instances in the various classes.

Class Index	Precision	Recall	F-Measure
1	79.740	77.211	78.455
2	80.909	84.891	82.852
3	78.992	82.563	80.738
4	80.159	64.952	71.758
5	64.633	27.739	38.818
6	66.850	56.574	61.285
7	81.194	80.285	80.737

Table 11: precision, recall and F-measure of classes using a neural network.

Table 11 shows detailed stats for each class. Precision values are quite good for all classes, with the maximum precision registered being 81% and minimum 64%. As for recall, its values are still quite different from each other but in general are better than what has been seen so far.

1	2	3	4	5	6	7	← classified as
77.21	20.98	0.05	0.00	0.13	0.03	1.60	1
13.18	84.89	0.80	0.00	0.41	0.58	0.14	2
0.00	8.55	82.56	0.88	0.08	7.93	0.00	3
0.00	0.00	25.29	64.95	0.00	9.75	0.00	4
6.22	62.60	2.68	0.00	27.74	0.76	0.00	5
0.03	16.55	26.38	0.45	0.02	56.57	0.00	6
17.69	2.02	0.00	0.00	0.00	0.00	80.29	7

Table 12: confusion matrix associated to neural network testing.

The resulting confusion matrix is shown in Table 12. We can see that instances of class 5 are still hard to recognize (as when using Naive Bayes), while classes 1,2,3 and 7 are classified correctly most of the time. The classifier is weaker on class 4 and 6, but can be considered a decent result in light of the small number of examples at its disposal.

5.3 Cross validation

To find out if 79.9% is a value for accuracy to be confident with, cross validation is executed. The resulting mean accuracy is 80.033%, with a 95% confidence interval of ± 0.274. Mean precision is 77.139%, mean recall is 67.837% and mean F-measure 70.848%. Also in this case, the cross validation indicates a mean accuracy which is nearly identical to the one in accuracy testing; furthermore, the confidence interval is small enough to guarantee that such value is very plausible. It can be seen in Figure 9 accuracies of each fold keep themselves quite near the mean with only a few of them noticeably far from it. Table 13 looks similar to Table 11, the one in accuracy testing, with a low recall value only for instances of class 5.

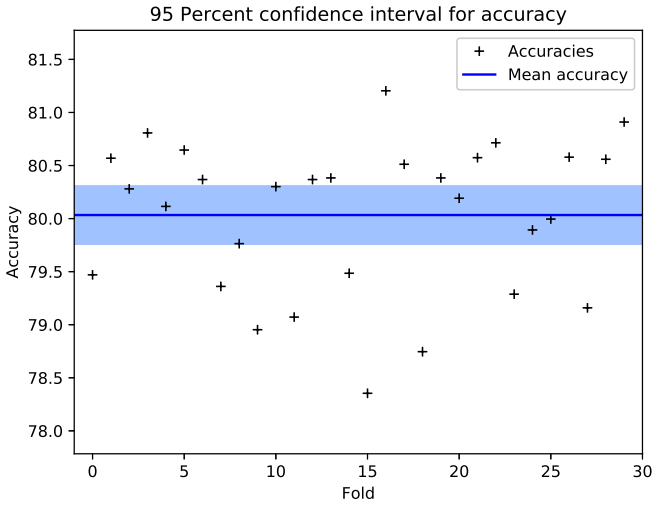

Figure 9: Scores obtained variating the number of hidden units in a neural network.

Class Index	**Precision**	**Recall**	**F-Measure**
1	79.744	77.588	78.556
2	81.134	84.981	82.963
3	78.641	84.546	81.423
4	81.413	69.292	74.716
5	68.007	27.227	38.370
6	66.303	53.938	59.130
7	84.729	77.285	80.773

Table 13: precision, recall and F-measure values resulting from cross validation.

Plotting the F-measure values results in Figure 10. As the bar chart shows, all classes except 5 present a high valued harmonic mean between precision and recall, around 80%. This confirms that neural networks are a powerful tool that can be used to achieve decent results in classification problems involving complex phenomena or a non optimal class distribution.

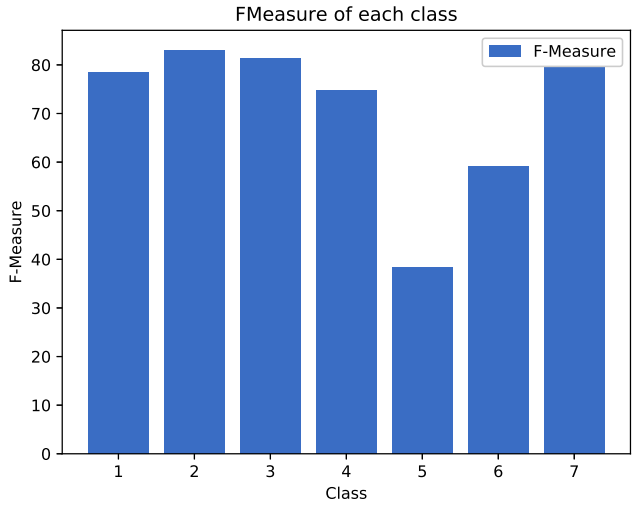

Figure 10: F-measure scores obtained using a neural network.

5.4 Learning curve

Dataset Size	**Accuracy**
58102	75.728
116203	76.140
174304	77.256
232405	78.099
290506	78.403
348607	79.264
406708	78.510
464809	78.318
522910	79.739
581011	80.050

Table 14: datasets size and relative accuracy using a neural network.

The last thing to understand about neural networks applied to the forest cover type problem is whether more data would have been helpful to achieve a better accuracy. The neural network is tested on the usual ten datasets of different size, Table 14 shows the collected values. At the beginning, with the smallest dataset, the accuracy is around 75%; then, the more instances are in the dataset, more accuracy grows. For the first time a quite noticeable linear dependency between dataset size and accuracy is observed, as can be seen in Figure 11. At the end an 80% of accuracy is reached, that is, a 5% increment.

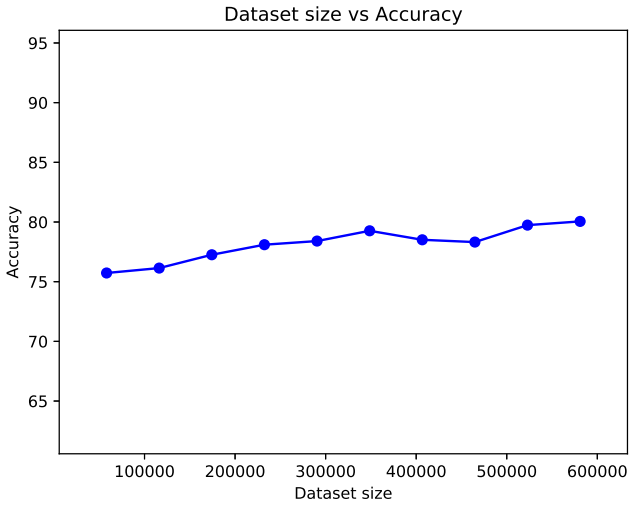

Figure 11: Learning curve for neural network.

6 Decision Tree

A decision tree is tool that uses a tree graph to model decisions and possible consequences. It is used in machine learning to find features-based rules according to which observations about an instance (represented in internal nodes) are mapped to a target class (the leaves). Decision Tree Learning works by recursively splitting the training set according to a chosen feature and its values; the choice of feature to split data in an internal node in the tree is crucial. Most popular characteristics observed to make the decision are *information gain*, that measures the quantity of information that a given feature carries and that is helpful in determining the label of the example, and *Gini impurity*, that is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset.

6.1 Parameters selection

In this subsection the most important parameters configurations are analyzed. The decision tree implementation in scikit-learn, the Python library used, the most important parameters that can be set are:

- *criterion*: the criterion that feature at an internal node is chosen with. Possible values are "gini", for Gini impurity, and "entropy", for Information Gain;
- *max_features*: the number of features to consider when looking for the best split. Tested values are $\{10, 15, 30, 35, 40\}$.
- *min_samples_split*: the minimum number of examples required to split an internal node. Tested values are $\{2, 5, 10, 30, 50\}$.

- *min_samples_leaf*: the minimum number of examples required to be a leaf node. Tested values are $\{5, 7, 15, 20, 50\}$.

The number of all configurations tested is 250. To perform the search, also in this case a subsample containing 7% of the original instances is used to speed up the operation. Figure 12 shows the score as a function of the configuration tested.

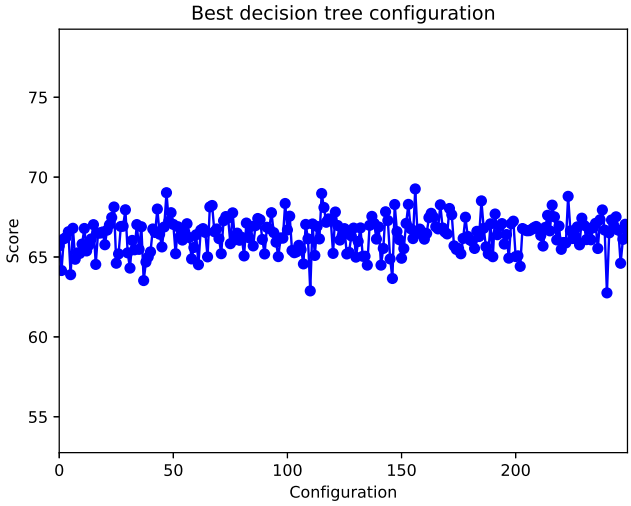

Figure 12: Optimal configuration search for decision tree.

The best score registered is 69% with *criterion* "gini", *min_samples_split* equal to 50, *max_features* equal to 15 and *min_samples_leaf* equal to 15. The choice of this configuration has brought a 5% improvement in accuracy according to this test.

6.2 Accuracy Testing

The decision tree classifier instance created with parameters found in precedence is run using the full dataset, with the usual 37% split for the test set. Its accuracy is equal to 86.864%, demonstrating that decision trees can be used to obtain pretty good results on the current classification problem. Now the attention is focused on precision and recall values for each class, shown in in Table 15. Decision trees seems to fit well in predicting the forest cover type, since precision and recall values are quite high, with maximum precision and recall registered equal to 89%. There is not too much distance between precision and recall values, in fact most F-measures values are noticeably over the 70% threshold. Table 16 shows the confusion matrix. Few misclassification problems are noticeable. Class 5 is recognized 55% of the times, while 35% of its instances are misclassified as belonging to class 2. In the case of class 4, 23% of its instances are classified as belonging to class 3. 19% of instances of class 6 are labeled with 3. Other values outside the diagonal are pretty low, so one can

deduce that this classifier has the potential to handle the prediction task at hand.

Class Index	**Precision**	**Recall**	**F-Measure**
1	86.863	86.379	86.620
2	88.315	89.741	89.023
3	83.586	86.391	84.965
4	75.492	70.697	73.016
5	71.246	55.524	62.410
6	74.338	69.098	71.622
7	89.644	84.911	87.213

Table 15: precision, recall, F-measure for decision tree.

1	2	3	4	5	6	7	← classified as
86.38	12.45	0.01	0.00	0.23	0.05	0.87	1
8.53	89.74	0.68	0.01	0.53	0.45	0.06	2
0.14	4.92	86.39	1.14	0.24	7.17	0.00	3
0.00	0.31	23.67	70.70	0.00	5.33	0.00	4
7.10	35.08	1.39	0.00	55.52	0.91	0.00	5
0.49	9.82	19.31	1.00	0.28	69.10	0.00	6
13.46	1.59	0.00	0.00	0.04	0.00	84.91	7

Table 16: confusion matrix of the decision tree.

6.3 Cross validation

Cross validation is performed to validate the decision tree instance used in accuracy testing. The resulting mean accuracy is 88.504%, with a 95% confidence interval of ±0.115. Mean precision is 83.836%, recall is 80.546% and F-measure is equal to 82.049%. It is an even better result than what has been obtained in accuracy testing. Figure 13 shows the confidence interval for accuracy computed using the accuracies of each fold. The maximum accuracy registered is around 89%, while the lowest is 87.8%; a difference of 1.2% that is small enough to explain the tight confidence interval around the mean. Table 17 shows precision, recall and f-measure values for each class. Precision and recall values are high, with class 7 having the maximum precision registered, 90%, and class 2 having the maximum recall registered, 90.8%. Class 5, the one most difficult to recognize, has decent precision and recall values.

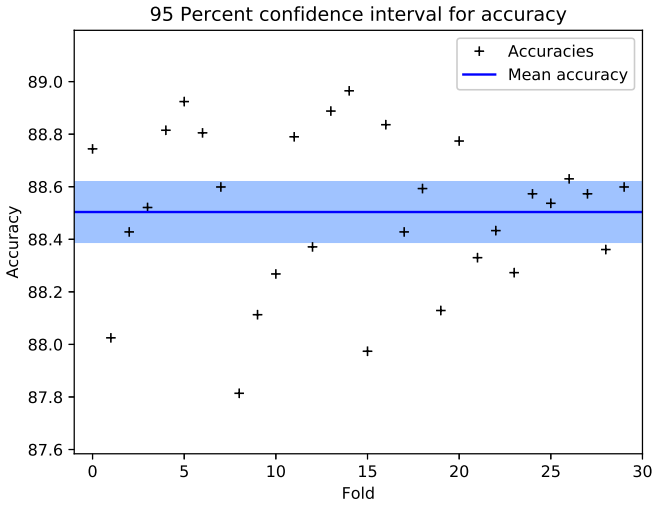

Figure 13: Accuracy interval for decision tree.

Class Index	**Precision**	**Recall**	**F-Measure**
1	88.444	88.043	88.242
2	89.768	90.864	90.312
3	86.364	88.400	87.365
4	79.202	72.558	75.601
5	74.595	62.550	67.987
6	78.429	74.835	76.568
7	90.054	86.572	88.267

Table 17: precision, recall and F-measure values resulting from decision tree.

Figure 14 shows graphically the F-measure values for each class. All the values are above 67%, with class 2 touching 90%. It follows that decision tree can indeed be a good classifier to use in this context.

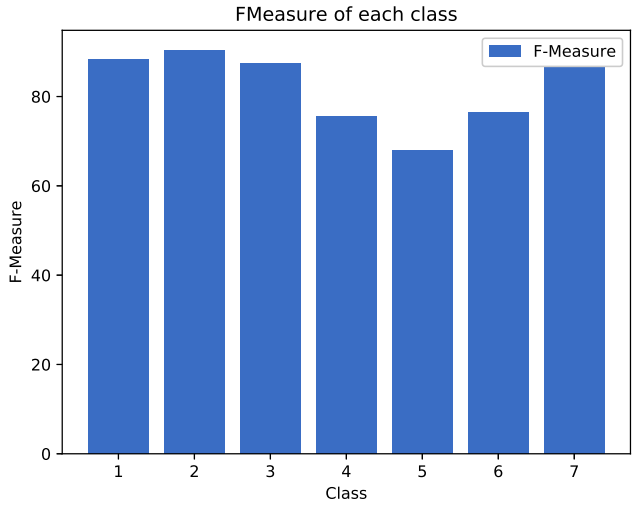

Figure 14: F-measure values for decision tree.

6.4 Learning curve

Now the learning curve will be plotted and analyzed. To do so, various accuracies relative to different datasets with different sizes are collected in Table 18. The accuracy relative to the smallest dataset is 76%, and the one relative to the greatest is 86%. It is a remarkable 10% increment in performance, that makes think about the possibility of even better performances with an greater dataset. Figure 15 shows this increment in a graphical way.

Dataset Size	**Accuracy**
58102	76.854
116203	79.703
174304	82.356
232405	82.532
290506	83.542
348607	84.368
406708	85.017
464809	85.746
522910	85.980
581011	86.327

Table 18: Datasets size and relative accuracies.

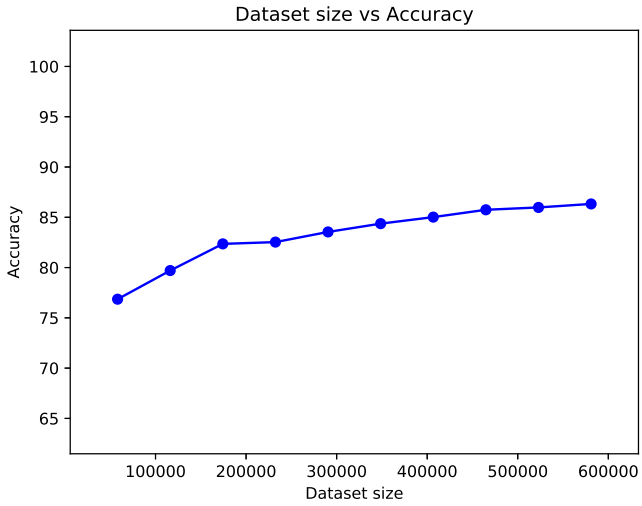

Figure 15: Learning curve of decision tree.

7 Random forest

Random Forest is an *ensemble method* used for classification and regression tasks. It operates by constructing a multitude of decision trees during the training phase and outputs the mode of the values given by individual trees. They are a powerful tool that contrast the overfitting tendency of a single decision tree.

7.1 Parameters selection

The parameters that can be set are the same as the decision tree ones plus the number of trees in the forest. That number is set to 10, that is the default one. The same configurations as in the decision tree case are tested (here repeated for convenience):

- *criterion*: the criterion that feature at an internal node is chosen with. Possible values are "gini", for Gini impurity, and "entropy", for Information Gain;
- *max_features*: the number of features to consider when looking for the best split. Tested values are $\{10, 15, 30, 35, 40\}$.
- *min_samples_split*: the minimum number of examples required to split an internal node. Tested values are $\{2, 5, 10, 30, 50\}$.
- *min_samples_leaf*: the minimum number of examples required to be a leaf node. Tested values are $\{5, 7, 15, 20, 50\}$.

The number of all configurations tested is 250. In this case the search is performed using a slighter smaller sample (5% of the original instances, because

it needs 10 times more time than a single decision tree). Figure 16 shows the score as a function of the configuration tested.

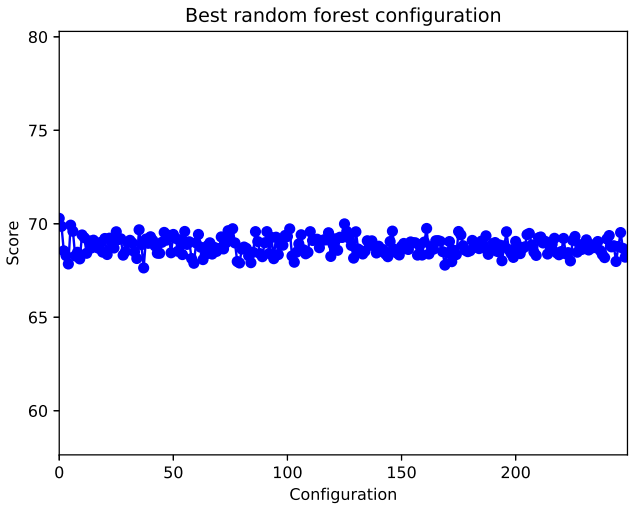

Figure 16: Optimal configuration search for random forest.

The best score registered is 70% with criterion "entropy", minimum examples split 2, max features 10 and minimum examples leaf equal to 5.

7.2 Accuracy testing

To get an idea about the performances of the newly created random forest instance, an accuracy test is run, with the usual split. The resulted accuracy is equal to 93.946%, an optimal result which, if confirmed by cross validation, could classify random forest as one of the best classifiers (the best seen so far) to solve the forest coverage type. Table 19 lists precision, recall and F-measure values for each class. Almost all values for the precision metric are above 90%, which means that the classifier is capable of recognizing all the classes really well. Recall values are high too, with class 5 having the lowest registered value of 71%.

Class Index	Precision	Recall	F-Measure
1	95.172	92.580	93.858
2	93.498	96.387	94.921
3	91.947	94.289	93.104
4	87.540	82.817	85.113
5	91.908	71.925	80.698
6	90.104	85.730	87.862
7	96.240	92.608	94.389

Table 19: precision, recall and F-measure values using random forest.

1	2	3	4	5	6	7	← classified as
92.58	7.05	0.00	0.00	0.04	0.02	0.31	1
2.97	96.39	0.27	0.00	0.17	0.17	0.03	2
0.02	2.22	94.29	0.55	0.07	2.86	0.00	3
0.00	0.00	14.89	82.82	0.00	2.30	0.00	4
1.70	23.89	1.89	0.00	71.93	0.59	0.00	5
0.14	4.42	9.03	0.65	0.03	85.73	0.00	6
6.69	0.69	0.00	0.00	0.01	0.00	92.61	7

Table 20: confusion matrix associated to random forest classifier.

Table 20 shows the confusion matrix. Almost all the instances are concentrated on the diagonal, except for 23.8% of class 5 instances being classified as belonging to class 2 and 14.8% of instances being classified with label 3.

7.3 Cross validation

To find the mean value of the underlying distribution of accuracy associated with the random forest classifier, with a confidence interval of 95%, cross validation is run. The resulting sample accuracy mean is 95.166%, and with a confidence of 95% the mean of the distribution is inside the confidence interval 95.166 ± 0.074. Also, mean precision is equal to 93.578%, recall is 90.232% and F-measure is equal to 91.773%.

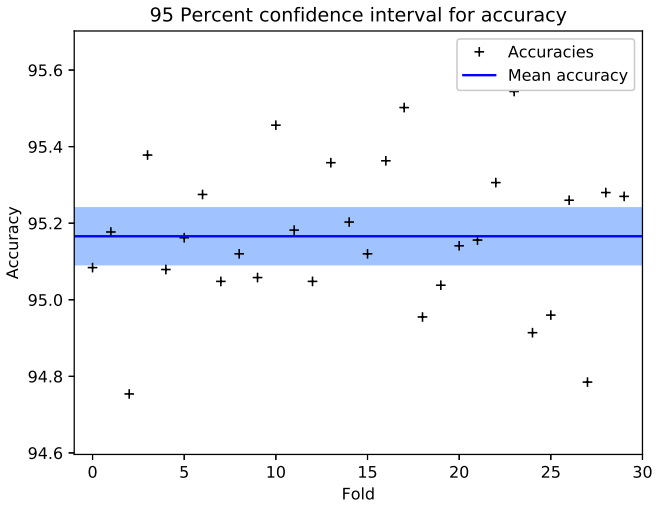

Figure 17: Mean accuracy confidence interval for random forest.

As can be seen in Figure 17, the confidence interval is really small, so the mean accuracy is expected to be nearly identical to the sample mean. This is because random forest performed very well on each fold, generating points in the Figure 17 that are close to each other.

Class Index	Precision	Recall	F-Measure
1	96.091	94.198	95.135
2	94.907	96.998	95.941
3	93.458	95.480	94.457
4	88.919	84.152	86.392
5	92.893	77.819	84.673
6	91.890	88.364	90.080
7	96.886	94.615	95.734

Table 21: precision, recall and F-measure values associated to random forest.

Table 21 shows that precision and recall values for each class are extremely high, even for class 5 where other classifiers have a noticeable drop in performance.

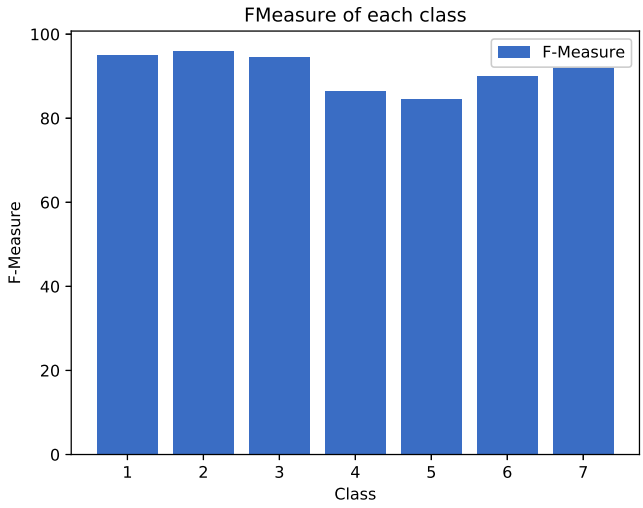

Figure 18: F-measure values resulting from cross validation.

Finally, Figure 18 illustrate the F-measure values associated to each class. The plot immediately remarks that random forest, because of its nature (that is, is an ensemble method), is capable of "memorizing" particular traits of certain instances that are rare in the dataset (and that other classifiers may lose due to the abundance of other type of instances). In fact, every class presents an high harmonic mean, meaning that random forest is well aware of the existence and nature of that class of instances.

7.4 Learning curve

The last part of the experiment on the random forest classifier concerns its ability to improve its performance in predicting instances when presented with a greater number of instances to use as training set. As usual, the dataset

is divided in ten parts of different sizes, and the classifier is tested on them. Table 22 sums up data collected during this test. With the smallest dataset the accuracy registered is 83.7%; with the largest the accuracy is equal to 94.1%. In other words a 11% increment in performance is registered. Furthermore, as it is clear in Figure 19, accuracy grows constantly as the dataset size increments. It can be deduced that random forest performance definitely depends on the dataset size.

Dataset Size	**Accuracy**
58102	83.710
116203	87.901
174304	89.323
232405	90.657
290506	91.491
348607	92.363
406708	92.724
464809	93.109
522910	93.527
581011	94.162

Table 22: Datasets size and relative accuracies.

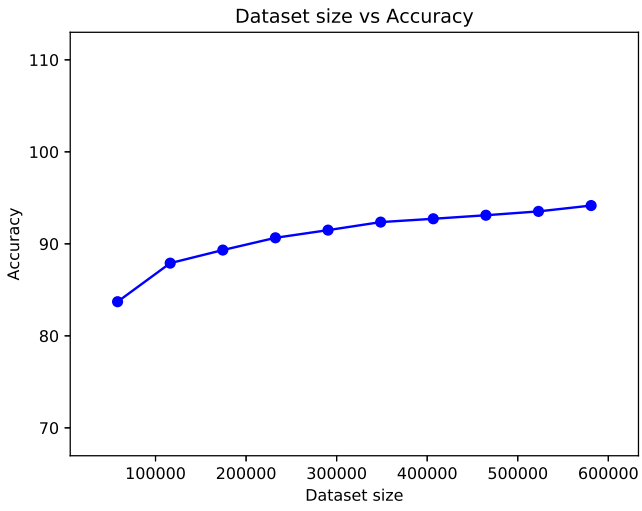

Figure 19: Random forest learning curve.

8 Models comparison

In this section data gathered in experiments relative to every tested classifier is used to perform a comparison between them in order to gain insights about

which one is more suitable to predict the forest cover type, why, and what are the difference with the other candidates. As first step classifiers' mean accuracies are plotted in Figure 20.

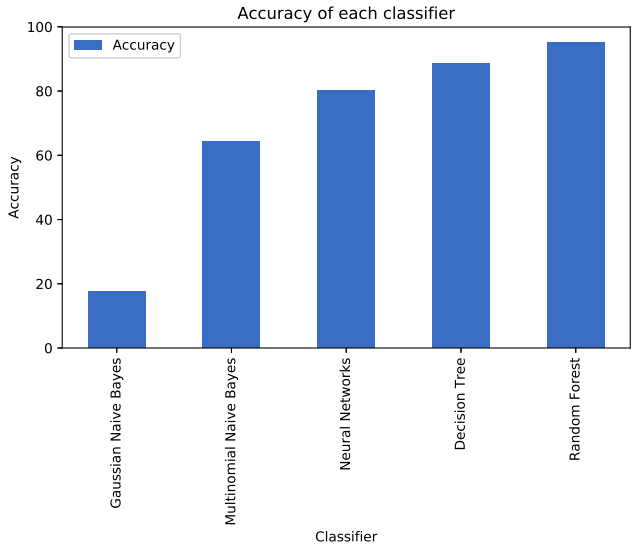

Figure 20: classifiers' accuracy.

The worst classifier is Gaussian Naive Bayes, with 17.46%, that means it is completely incapable of modeling the phenomenon. Next comes Multinomial Naive Bayes, which scores 64.1%. An increment in accuracy equal to 46.6%, obtained just by switching the underlying distribution of feature values. The Neural Network trained obtains an accuracy of 80%, classifying itself as a potential solution. Decision Tree has a score of 88.5% but Random Forest is the best classifier with an accuracy of 95.1%

In general can we stated that the feature independence assumption is not valid (this is quite obvious, since various features encodes single attributes), and this is why Naive Bayes does not perform well.

Neural network classifier has fairly good precision and recall values (Table 13) for the most frequent classes, but experiences a drop in performances on instances belonging to less frequent classes, especially class 5. This may be due to the nature of the training algorithm, since it gives more importance to latest items picked from the training set, and the probability of them to belong to class 1, 2, 3 is (a lot) higher than belonging to the remaining ones (see Figure 1).

Decision tree classifier goes a step further gaining a good accuracy value within a tight confidence interval and good F-measure values for each class (Figure 14). Decision tree outperforms Neural Networks and Naive Bayes because its structure matches very well the type of features that describe a generic instance of the dataset. Most of the features are binary values and can be interpreted as answer to a "binary question" when passing trough an internal node of the tree in order to split the data (in training phase) or to classify an instance.

Finally, Random Forest has on this problem the benefits of the decision tree plus the fact that it has multiple different decision trees, a subset of which potentially focusing on a particular class. This can explain the high accuracy of this classifier and its good performance even on class 5 instances that are the most difficult to recognize.

The last words are on the learning curve of the classifiers. Neural network, decision tree and random forest's accuracy show dependency on the number of samples in the dataset. The dataset is quite large to build a good classifier as it has been seen, but more data could further improve accuracy and make error rate negligible.

In conclusion, this report has seen various classifiers trying to model the forest cover phenomenon. After a series of tests and a comparison Random Forest tool classifies itself as the one to be chosen to solve the problem, especially when class distribution is highly unbalanced.