Natural Computing, Assignment 3

Dennis Verheijden - s4455770 Pauline Lauron - s1016609
Joost Besseling - s4796799

March 29, 2018

Combining, Bagging & Random Forests

1

(a)

- The probability that all three doctors give the correct answer is $0.8^3 = 0.512$.
- The probability that exactly 2 doctors make the right call is $0.8 * 0.8 * 0.2 + 0.8 * 0.2 * 0.8 + 0.2 * 0.8 * 0.8 = 0.384$. Therefore, the probability that *at least* two doctors make the right call is $0.512 + 0.384 = 0.896$.
- The probability that this group makes the right decision based on majority voting is $0.512 + 0.384 = 0.896$ since the majority is when there are at least two individuals.

(b)

The general formula is

$$P(\text{correct predictions} > c/2) \sum_{i=\lfloor n/2 \rfloor}^{n} p^i (1-p)^{n-i} \binom{n}{i}.$$

Using this formula, we find a probability of about 0.826.

(c)

If we use 10000 runs of the simulations, we get an approximately equal result of 0.826. Writing out more decimals gives us a difference of 0.0057. So our approximation is pretty good.

(d)

We decided to use a surfplot for the visualization. Here we can easily spot the differences when variables change relative to each other.

The surfplot can be found in figure 1. What we can observe from this plot is that the jury size matters for a low number of people, but that this effect has exponentially diminishing returns. The competence, as expected, has the highest effect on the probability of correctly making the right decision.

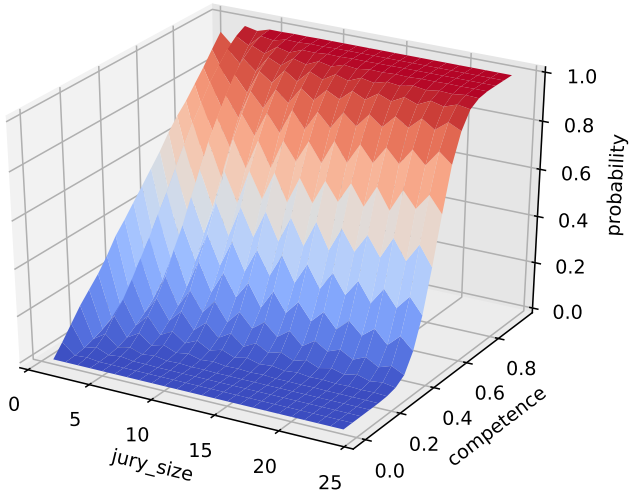

Figure 1: Surfplot of the probabilities as a function of the jury size c and competence p.

(e)

The probabilities for making the correct decision for the groups are:

- **radiologists**: 0.850
- **doctors**: 0.896
- **students**: 0.826

To reach the same probability for making the correct decision as the group of doctors, using only students. You would need 28 students. These would, collectively, have a probability of making the correct decision of 0.898.

2

The filled in table may be found below in table 1. Here we can see the values for different combinators. Colors denote the choice that the classifier will make. What we can note from this table is that the choices of these classifiers are in most cases the same.

$p_1(\omega\|x)$		$p_2(\omega\|x)$		$p_3(\omega\|x)$		Mean		Max		Min		Prod	
A	B	A	B	A	B	A	B	A	B	A	B	A	B
0.9	0.1	0.9	0.1	0.0	1.0	0.6	0.4	0.9	1.0	0.0	0.1	0.0	0.01
0.9	0.1	0.9	0.1	0.3	0.7	0.7	0.3	0.9	0.7	0.3	0.1	0.243	0.007
0.9	0.1	0.2	0.8	0.1	0.9	0.4	0.6	0.9	0.9	0.1	0.1	0.018	0.072
0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0

Table 1: Classifier Combination for different methods. Values indicated in orange denote the decision the combiner would make. Values indicated in red denote a tie, the choice here is based on the implementation.

3

The formula for the percentage of items that are not in a sample using Bootstrapping is:

$$p_{notPresent} = (1 - \frac{1}{N})^N$$

We also made simulations which approximate this function, as found in figure 2. Here we see that our simulations approximate our formula. Which adds evidence to its correctness.

Now that we have this implementation, we can also make simulations for different N. The results for this are shown in figure 3. Here we can see the comparisons for different N. We see that the probability of items that do not occur converge to 0.37 for larger N. We see that there probably is an exponential relation as the probability first rapidly increases to 0.25 for $N = 2$ and at $N = 5$ it is almost at convergence.

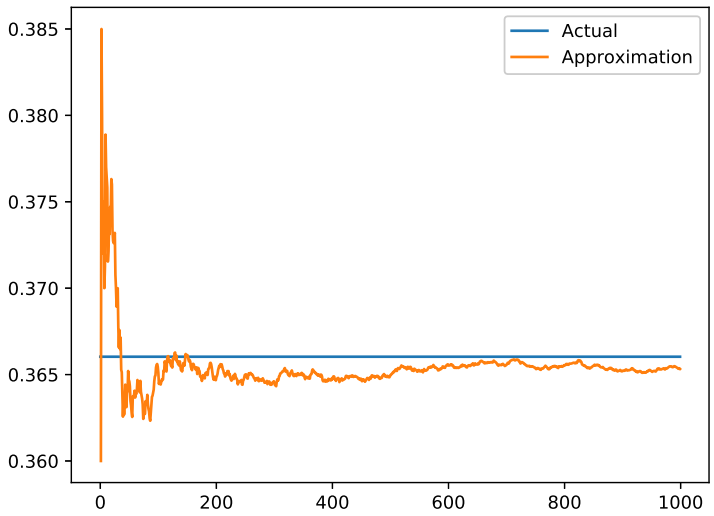

Figure 2: Comparison of our simulation and the exact value.

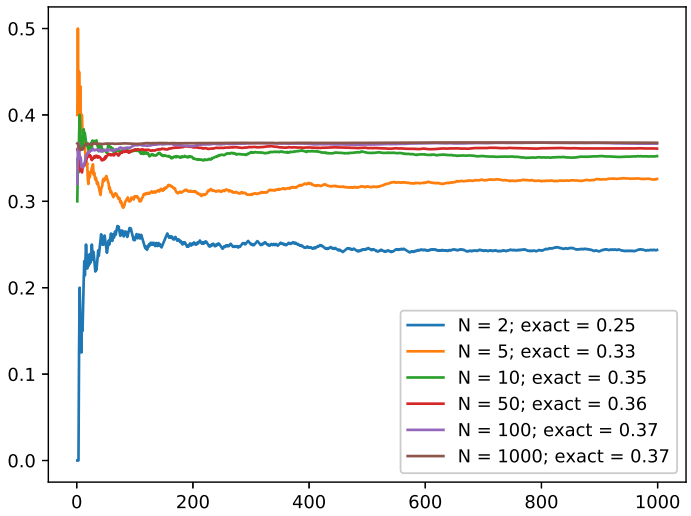

Figure 3: Comparison of different sample sizes for the percentage of left-out items.

4

The difference between bootstrapping and random subspaces is how the training data is constructed.

- In bootstrapping, you randomly select data from the whole dataset with replacement. Such that you can have samples that occur more than once.
- In random subspaces, you randomly select features from the whole dataset. So instead of randomly drawing samples, you draw features from these samples.

5

For random forests, the importance of a variable is calculated using a criterion. The most used measure in Forest Classifiers is the Gini Impurity. In short, the Gini Impurity looks for every feature, if we would split based on this feature, how pure the split would be, i.e. how well it separates the classes. The main drawback of this approach is that these classifiers are extremely sensitive to small changes in your training set. One slight change could create a drastically different tree.

6

For this experiment we used the handwritten digits, aka MNIST, dataset. As this dataset is easy to interpret and it has a decent difficulty to learn as the dataset contains 64 features and 10 classes.

We used the standard `RandomForestClassifier` from the python module `sklearn`. We found two variables that we thought to be important: *max_depth*, *n_estimators* and *max_features*. These respectively mean the maximum depth of a tree, the number of trees that are trained in the forest and the maximum number of features Random Forest is allowed to try in individual tree. The result of manipulating the first two parameters are found in figure 4.

What we can note from this figure, is that both parameters converge. The default value for *n_estimators* is 10. This value is roughly the value for which the algorithm converges. The default value for *max_depth* is not set, i.e. it expands until all leaves are pure or contain less than 2 samples. To better see whether setting this value is helpful, we can examine the results for certain values of the *max_depth* while leaving the parameter *n_estimators* to the default value. The results may be found in figure 5. What we can observe is that for a max tree depth of 1 or 2 the RandomForestClassifier is already way above chance, being 0.1. The performance peaks at a max tree depth of 10. After this it starts to decline. This is because the tree is overfitting on the training set, which is an easy pitfall of this type of classifier. However, since it is an ensemble, the damage of overfitting is limited. If we increase the max_features, we supposed to have a better performance. However, if we used to much features, this will decrease the diversity of individual tree and we want don't want dependence between trees to have l tree in order to have better performance.

To test different combination of parameters, we use the Grid Search. We obtain a score around 0.96 with using the logarithm function for the max_features, 100 for the max_depth and 40 for the n_estimators.

Conclusion, setting the parameter *n_estimators* might not be necessary, depending on the domain. However, more estimators lead to more stable predictions. The parameter *max_depth* however, should be carefully set, as it has the most effect on the mean accuracy of the RandomForestClassifier.

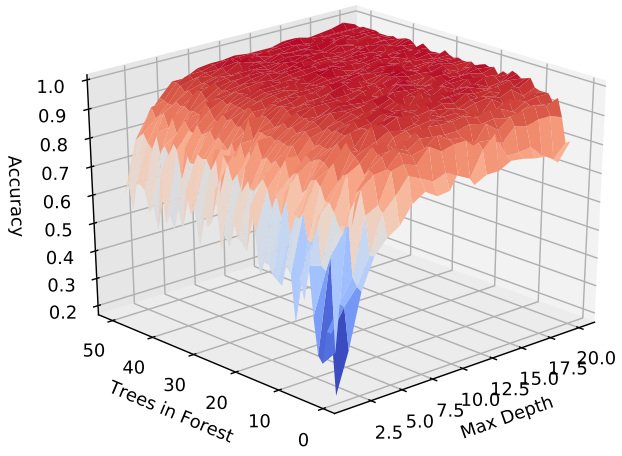

Figure 4: The mean accuracy of RandomForstClassifiers for different parameter setups.

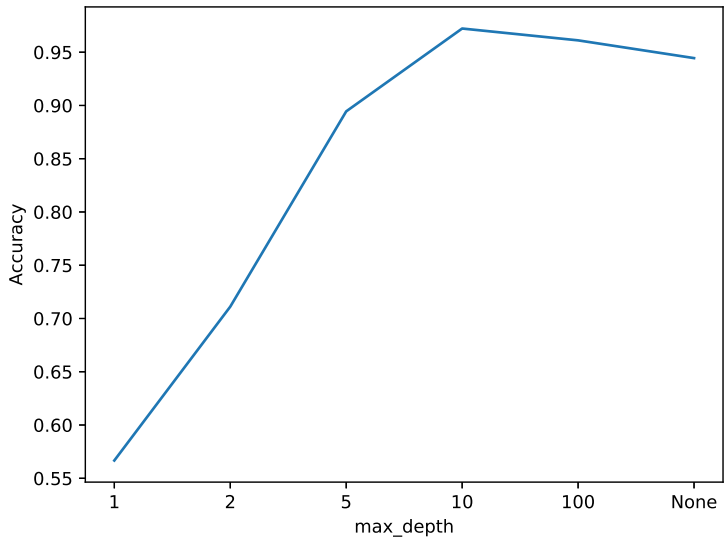

Figure 5: The mean accuracy of RandomForstClassifiers for choices for the max depth of the trees. The number of trees is set to the default value of 10. A tree depth of *None* means that the tree is allowed to split until the leaf nodes are either pure or have less than 2 samples.

Boosting

1

2

3

In bagging we train multiple learners by sampling from the dataset with replacement. We combine them by, for example, taking the average decision, or using a majority vote.

In boosting, we also train multiple learners. But we use the learners to weight the samples from the dataset, such that samples that we predict wrong early on, get a higher wait. We combine the learners similarly to the boosting case, but we weight them according to their results on the training set.

4

5