Forest Cover type Classification

Project of Statistical methods for machine learning

Sara Ramazzotti - 960893 - sara.ramazzotti@studenti.unimi.it

A.Y. 2020-21

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

1 Assignment

Download the Cover Type Dataset [Fct] for multiclass classification. Implement AdaBoost from scratch and run it using decision stumps (binary classification rules based on single features) as base classifiers to train seven binary classifiers, one for each of the seven classes (one-vs-all encoding). Use external cross-validation to evaluate the multiclass classification performance (zero-one loss) for different values of the number T of AdaBoost rounds. In order to use the seven binary classifiers to make multiclass predictions, use the fact that binary classifiers trained by AdaBoost have the form $h(x) = sgn(g(x))$ and predict using $argmax_i g_i(x)$ where $g(x)$ corresponds to the binary classifier for class i.

2 Introduction

2.1 Analysis of the dataset

The *Forest Cover Type Dataset* presents the observations collected from each landmark of the Roosevelt National Forest in Colorado. It has 55 columns, where the first 54 are the features of the land considered and the last one indicates the preferred *cover type*. It presents 581012 samples [Fct]. Data from the first ten columns are numerical values $\in \mathbb{Z}$; the remaining represents binary features $\in \{0, 1\}$.

The last column, as said, indicates the expected cover type classification. There are seven type of covers and they are indicated as numbers in the set $\{1, 2, 3, 4, 5, 6, 7\}$. It has

to be noticed that the dataset is not balanced, for each class there are different numbers of samples. We can see the distribution of the labels in the following histogram:

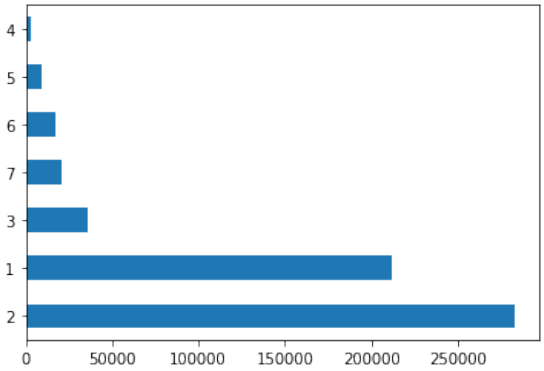

Figure 1: Cover type distribution. On the ordinate axis there are the seven classes, in ascending order w.r.t. the number of samples on the abscissa axis.

Luckily the raw data does not present null values, so the dataset does not need manipulation. The distribution of each feature can be found at Kaggle's page.

2.2 Adaboost approach

2.2.1 Week learners used

Adaboost is an ensemble method used to achieve a bias-variance tradeoff better than the one achieved by the single combined predictors that are used by the algorithm. Adaboost can train very simple weak learners, such as decision stumps. Decision stumps are classifiers of the form $h_{i,\tau} : \mathbb{R}^d \to \{-1, 1\}$ defined by $h_{i,\tau} = sgn(x_i - \tau) \cdot b$, where $i = 1...d$, $\tau \in \mathbb{R}$ and $b \in -1, 1$. In the implementation b is chosen as $+1$ for simplicity [CB]. Note that d is the number of features of the samples, in our case 54, and τ is the number that permits to split the samples in two groups: one labeled with $+1$ and the other with -1. Obviously the labels predicted in such a way are not perfectly labeled, but we can chose τ and i in order to pick in the set of all possible couples of (i, τ) the one that generates the predictor with the minimum weighted training error (ERMDS [SSS14]). This approach permits to include in the train phase a weight distribution of the samples, useful for Adaboost.

2.2.2 Multiclass classification

The base version of Adaboost is designed for binary classification. As previously mentioned, our dataset presents seven different classes. A way to apply Adaboost with decision stumps on this dataset is to train seven different adaboost classifiers and then combine the results using $argmax_i g_i(x)$ to predict the labels, where $g(x)$ corresponds to the binary classifier for class i. Each predictor is trained with the dataset modified using the encoding *one-vs-all*: i.e. for class *2* we mark as $+1$ the samples classified as 2, and as

-1 the samples classified as 1, 3, 4, 5, 6 or 7. In such a way each predictor tries to predict if the example belongs to that particular class or to another one. Moreover, doing this kind of approach it can be studied the behaviour of adaboost for each type of sample. The idea is that maybe it is easier to classify some classes with respect to others and the class that is predicted with more difficulty could affect the entire multiclass prediction.

At this point I can present the pseudocode of the approach adopted. I need to feed into the model the entire training set untouched. Then the fit phase maps the given set to the form one-vs-all one class at a time. For each training set mapped this way, the classic adaboost can be executed. At the end of the train phase, for each class the model will have memorized the weak predictors trained at each round of the algorithm, their weights and their errors (useful for the analysis). These groups of predictors-weights are then used to form the seven binary classifiers of our interest, in the form $f = sgn(\sum_{n=1}^{T} w_i h_i)$. Finally they are grouped together to get the multiclass prediction: $argmax_i f_i$.

Algorithm 1 Pseudocode of the multiclassifier Adaboost

```
binaryPredictors ← []
for k in classes do
    dataset ← map(dataset, k)
    predictor ← adaboost(dataset, T)
    add predictor to binaryPredictors
end for
return predict with argmax_i binaryPredictors_i
```

3 Analysis of the output

In this section I want to show the results of the cross validation on my implementation of Adaboost. The cross validation was made on 5 folds, using the implementation of KFold from *scikit learn* [Kfo]. The main goal was the evaluation of the algorithm at the increasing of rounds of adaboost. For better timing performance two decisions were made:

- first of all, the python notebook that execute the cross validation was runned with google colab;
- secondly, two experiments were made using two subsamples of the entire dataset.

The first subsample considered is built with a random draw of one thousand examples from the original dataset. This small group of data permits to run the cross validation more quickly, to use a quite large number of rounds (35 is the chosen number) and to better understand the trend of the errors. It can be seen in the top-left subfigure of Fig. 2 that the algorithm does not behave so bad and that the errors tend to decrease even if not so quickly apparently.

As we know from the theory [CB] the training error of adaboost is upper bounded by $exp(-2\sum_{i=1}^{T}\gamma_i^2)$ where $\gamma_i = 0.5 - \epsilon_i$ and ϵ_i is the weighted training error of the weak

learner of round i. So the training error should go down exponentially with respect to T. I can verify this saving the errors of the weak classifiers. In Fig. 3 and 4 are represented in the first column the upper bounds and the training error for each single class considered, and in the second column the trend of the errors ϵ_i. Those results are taken from the last fold that have fitted the model. As you can see the trend of the training errors tends to be under the upper bound at every class and in particular 0 is reached by the class 4, around the 18th round. Even classes 3, 5, 6, and 7 almost reach the zeros. Instead classes 1 and 2 have the zeros quite far from their actual error state. We can see in the graphs of Fig. 2 the results of the cross validation for each one-vs-all study, which confirms the trends just discussed. For cover types 1 and 2 the errors goes down not so quickly, as expected. A possible reason of these facts is that there are not sufficient data to discriminate the two more present groups of the dataset (recall Fig. 1) with respect to the other four or between them. Beside class 4 for which the zero is reached, in the other classes even if it is not clear looking at the lines that the errors are reaching the zero, it can be seen in the ordinate axis that the errors are really small and quite near the zero. Moreover the test errors too are really low so for our dataset the data characterizing the samples of the classes with less samples are really well used by our stumps.

The second group of data considered has a grater number of samples, one hundred thousand. Applying cross validation on this set is much more expensive in terms of time complexity so I reduced the number of rounds assessed. We know that this implementation of adaboost is efficient, because the fitting time of the decision stump is efficient in the order of $\mathcal{O}(md)$ where d is the number of features (54) and m the number of examples. Though this process is efficient it has to be noticed that if m is a big number it still takes a lot of time, considering that we need to train a different weak learner at every round of adaboost and that in this particular implementation we need seven of them. Moreover during the cross validation we need to repeat the process for every training group.

The results do not differ too much from those of the previous experiment. It confirms the hypothesis of some underling schema of the observations that tend to make difficult the recognition of classes 1 and 2 alone as can be seen in Fig. 6. On the other hand It can be noticed in graphs of Fig. 5 that with this group the difference between test and training errors is even closer than the previous observation, which means that the characteristics of the test and train sets considered in CV are reasonably quite the same.

4 Conclusions

In this study the implementation follows a naive approach to multiclassification, exploiting the fact that the binary predictors are of the form sgn and that when they are combined they can predict by majority the true label. Another approach, which would have speeded up the analysis, is to use an adaboost algorithm specifically studied for multiclassification [YF14].

Figure 2: Results of CV on the set of 1000 examples. Means of Training and Test errors at the different stage of T. In the top-left subfigure is represented the true resulting classification and in the others subfigures it can be seen the error of the single class classification. Areas with a shaded color indicate the standard deviation at that point.

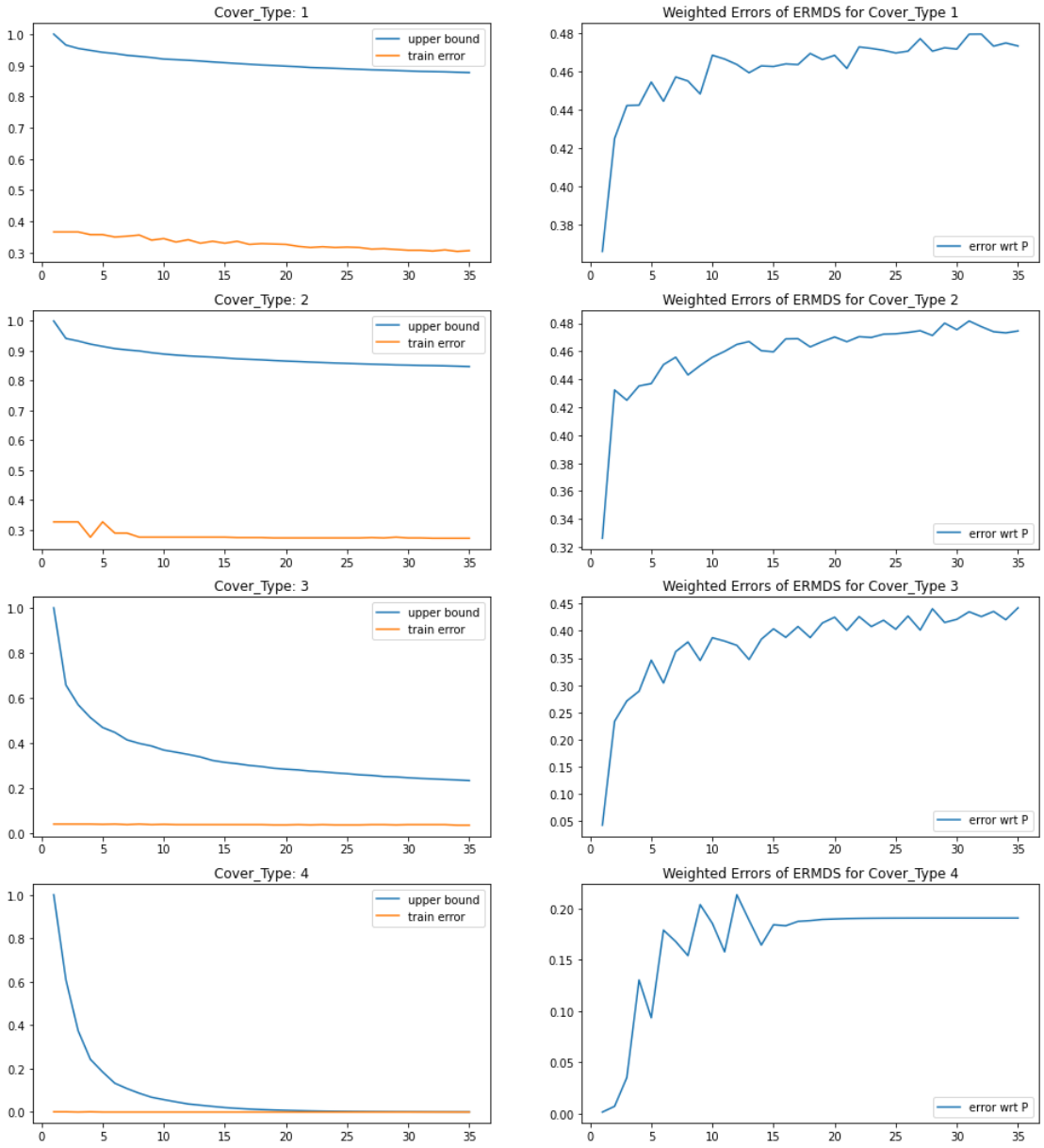

Figure 3: How the chosen weak learners for the last fold behave with respect to training error. Sample of 1000 examples.

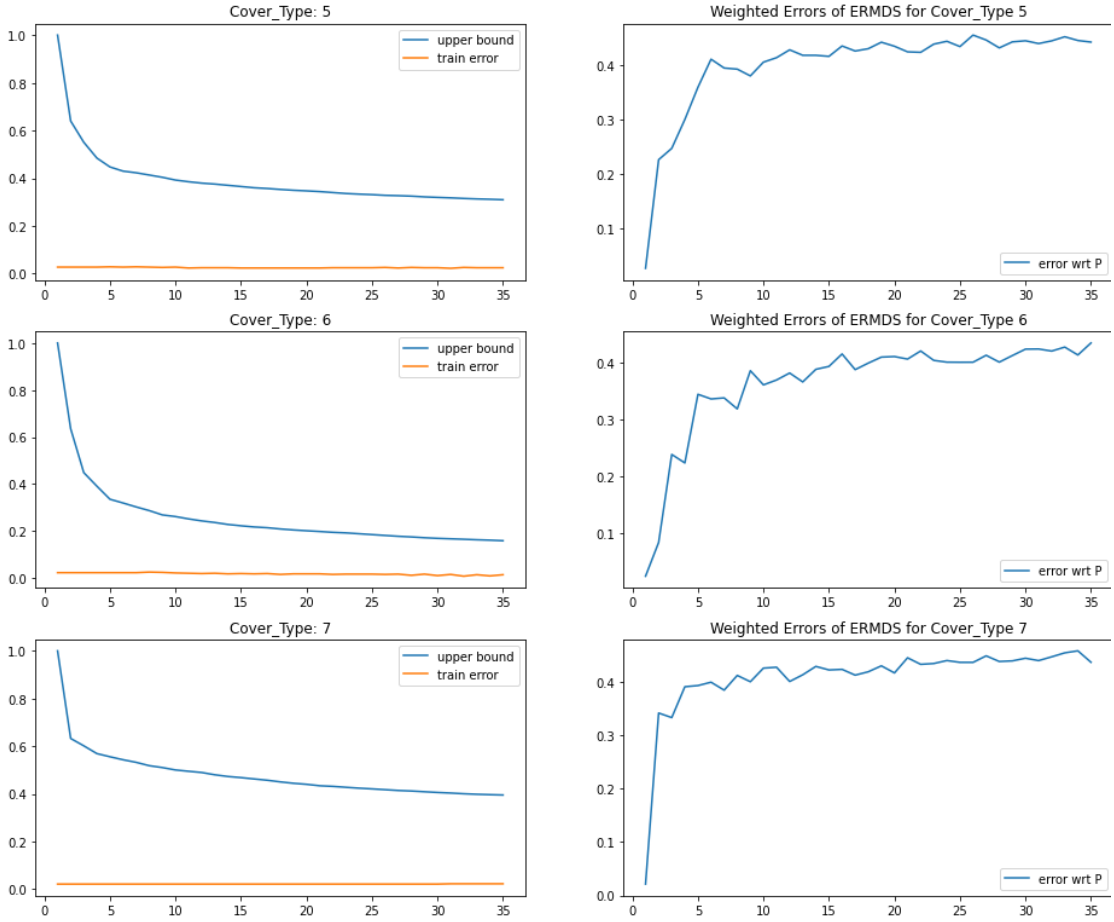

Figure 4: Cont. Fig. 3

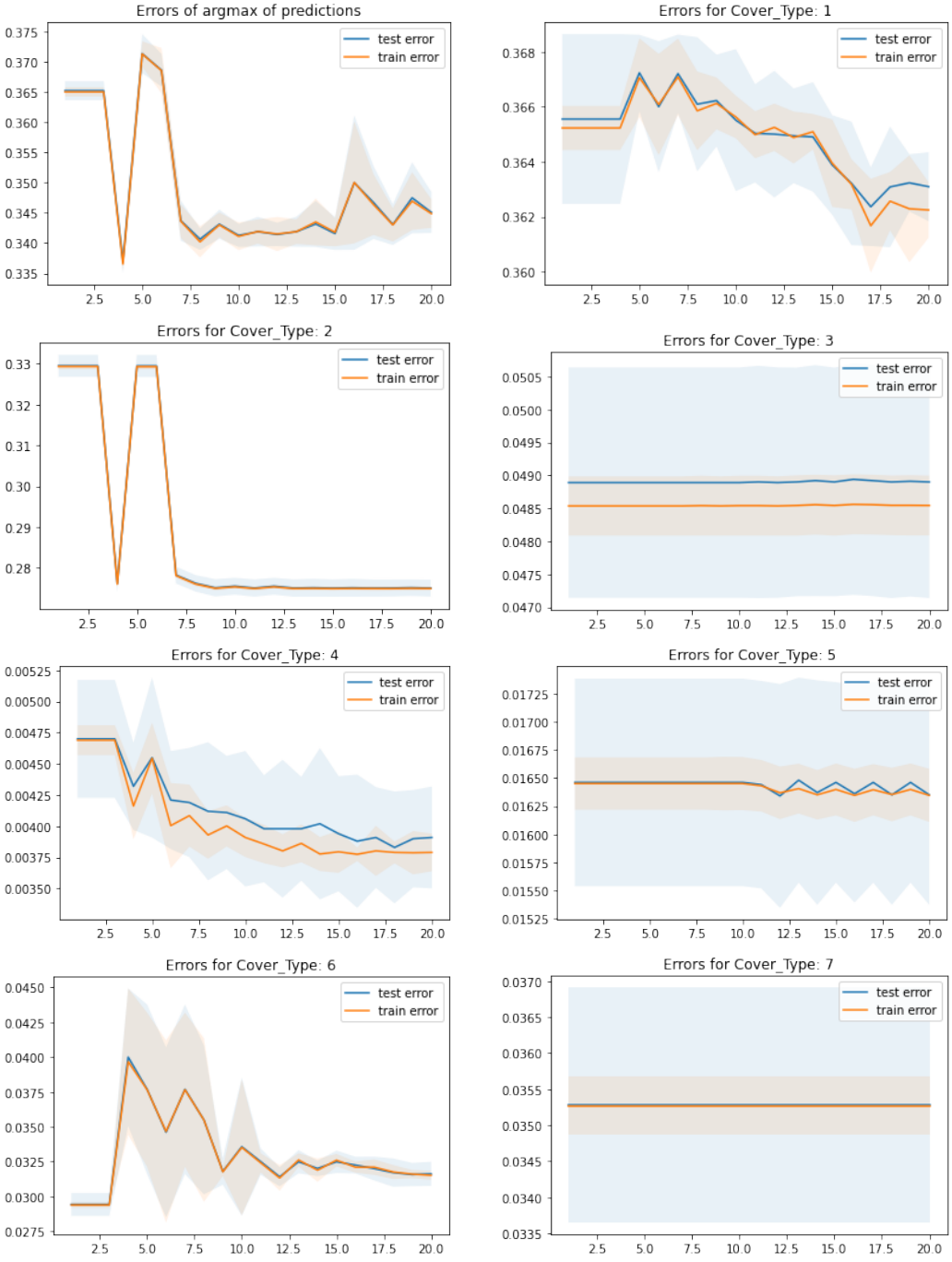

Figure 5: Same description as Fig. 2 for the bigger dataset.

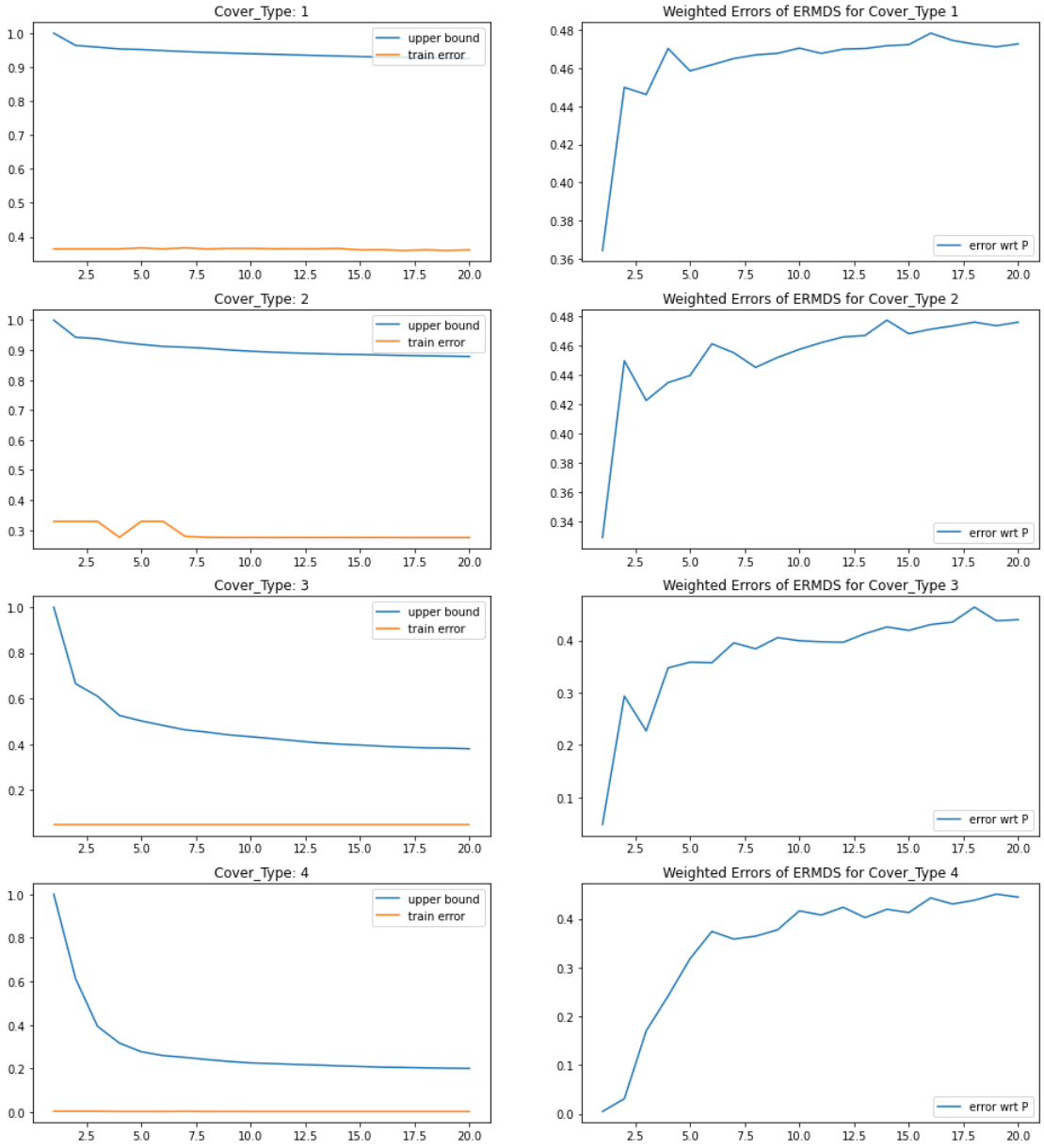

Figure 6: Same description as Fig. 3 for the bigger dataset.

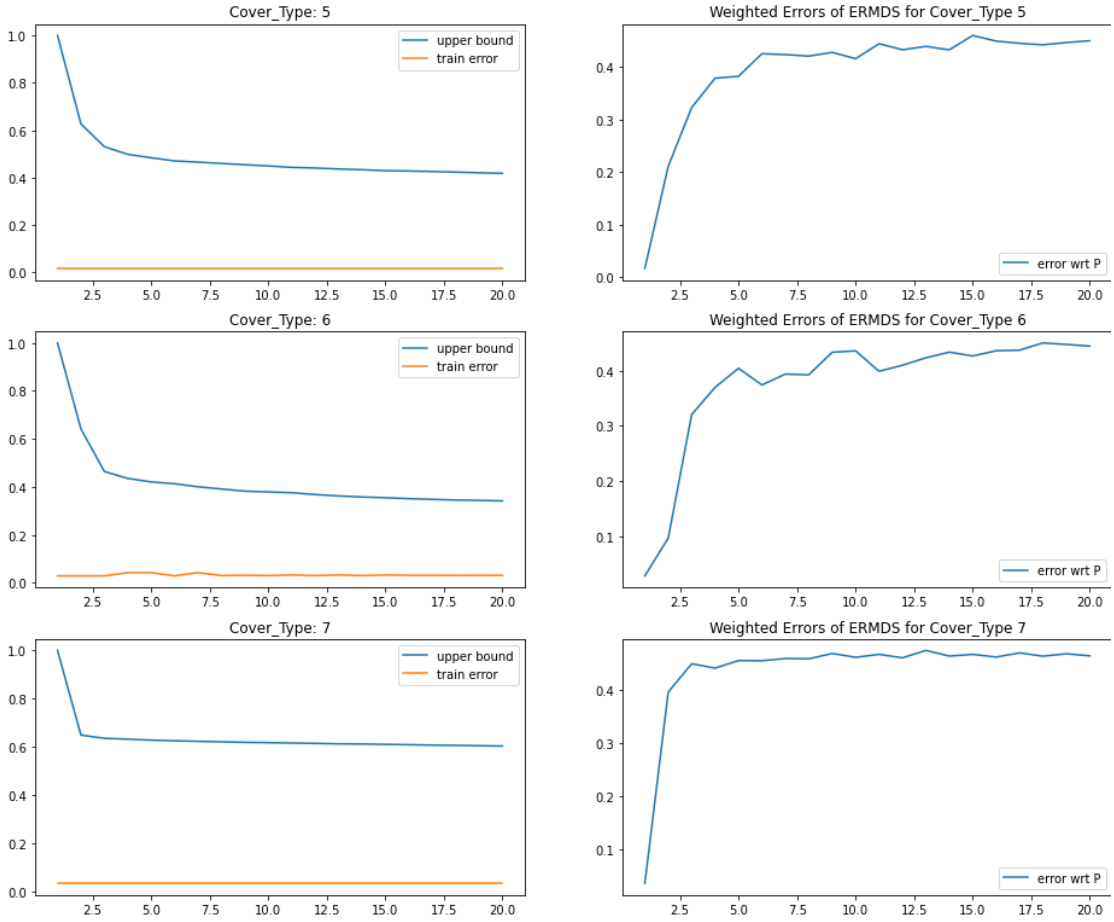

Figure 7: Cont. Fig. 6

References

[SSS14] Shai Ben-David Shai Shalev-Shwartz. "Understanding Machine Learning: From Theory to Algorithms". In: Cambridge University Press, 2014. Chap. 10.1.

[YF14] Robert Schapire Yoav Freund. "Boosting: Foundations and Algorithms". In: The MIT Press; Illustrated edition, 2014. Chap. 10.

[CB] Nicolò Cesa-Bianchi. *Boosting and ensemble methods*. URL: `https://cesa-bianchi.di.unimi.it/MSA/Notes/boosting.pdf`. accessed: 19.10.2021.

[Fct] *Forest Cover Type Dataset*. URL: `https://www.kaggle.com/uciml/forest-cover-type-dataset`. accessed: 19.10.2021.

[Kfo] *sklearn.model*$_s$*election.KFold*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html`. accessed: 20.10.2021.