AdaBoost
IT3105

Nordmoen, Jørgen H.
Østensen, Trond

November 20, 2012

Abstract

In this paper we will describe how we implemented AdaBoost, two weak classifiers and what results we got on a number of datasets. We will describe the datasets and why we chose them. We will explain how we divided those datasets into buckets for classification and how that is supported in our implementation. We will end this paper with a discussion of our results trying to explain and illustrate why.

Contents

List of Tables

1 Implementation

In this section we will talk a bit about our implementation. We will describe how it manages to classify new instances, how we generate new classifiers and how it can be extended without changing any implementation.

From our AdaBoost implementations view all we generate are Hypothesis[1] which comes from Generators[2]. These interfaces dictates what all new classifiers must support. A Generator creates a new Hypothesis with the basis in weights and the training set. An Hypothesis only needs to support setting its weight and classifying a new instance[3].

From an Hypothesis point of view all data is of type DataElement[4] which contains a number of attributes specified when it's created. It also contains a classification which it must be able to produce. DataElement uses Java generics to support different types of attributes and classifications which is determined by the filter used. Each Generator must support the same attributes and classifications as the DataElement used, but this is only checked at runtime.

Our data flow starts with reading all lines in a file creating DataElements with String values. Then we apply a filter which must implement the interface Filter[5]. After that is done we shuffle the dataset using a specified random seed to be able to replicate our tests. We then split the dataset into a training set and test set with a split specified on the command-line. After that we start AdaBoost and try to classify the test set.

The meat of our implementation lies in the Generator class which dictates how a specific classifier is created. This is the class which is defined on the command-line and this is the class which can be configured from the command-line.

This class must be able to receive a list of options where it can act on each option according to its own wishes which is done so we can configure each Generator, this is among other things where we pass the depth parameter to the Decision Tree Generator.

Each Generator class uses its own logic to create a classifier of the correct type based on the given data. This means that AdaBoost does not know nor care what a Generator creates as long as it implements the Hypothesis interface. This also means that our implementation is very general. When creating a new classifier type one has to implement a new Generator and a new Hypothesis class, that is all. To support new datasets one only has to implement a new filter to convert from String to a specific type. This was done to decouple the datasets from the classifiers, but it also means that we have to implement less code since each classifier shares the filters.

The filters are tasked with both converting the String values read from a file to the correct format, but it's also tasked with creating buckets for the classifiers. With this we mean that the filter must split the values in each file into proper buckets since our classifiers can't handle large ranges. This is also the reason why we have a filter per dataset. Since we had a hard time creating

[1] no.ntnu.ai.hypothesis.Hypothesis

[2] no.ntnu.ai.hypothesis.Generator

[3] We have created an abstract class for convenience, situated in no.ntnu.ai.classifiers.Classifier

[4] no.ntnu.ai.data.DataElement

[5] no.ntnu.ai.filter.Filter

an algorithm for splitting the ranges in different datasets we chose to have it like this for ease and to be able to control the buckets in more detail.

2 Results

Below is all the results from all our runs. We have used a couple standard values to control the randomness and the split of the datasets.

For each run we have used a split of "0.2" which means that we use 0.2 of our dataset as a test set. We have also used the random seed "42" for all runs this means that in every place where we need a random number the generator for that number is initialized in the same way with the same number each time. This creates reproducibility and ensures that we can be certain that our results are given the same starting point every time.

For each dataset we will point out what is needed to replicate the results.

The classifier options below does not conform to our actual command-line style which is down to space requirement. To use just add the sentence ' –classifier no.ntnu.ai[nbc—dtc].' in front.

2.1 Glass Dataset

This dataset represents the classification of different glasses in a crime scene.

To run, use these options with the classifier options wanted.

Listing 1: Glass dataset general options

```
--global 0.2 42 --file glass.txt --filter no.ntnu.ai.
    filter.GlassFilter
```

NBC #	Training Error	Standard Deviation	DTC #	Training Error	Standard Deviation	Test Error	Classifier option
1	0.339	N/A	0	N/A	N/A	18/43(41%)	NBCGenerator 1
0	N/A	N/A	1	0.181	N/A	17/43(39%)	DTCGenerator 1
5	0.391	0.045	0	N/A	N/A	19/43(44%)	NBCGenerator 5
10	0.371	0.042	0	N/A	N/A	19/43(44%)	NBCGenerator 10
20	0.358	0.034	0	N/A	N/A	20/43(46%)	NBCGenerator 20
0	N/A	N/A	5	0.256	0.043	18/43(41%)	DTCGenerator 5
0	N/A	N/A	10	0.673	0.133	23/43(53%)	DTCGenerator 10 1
0	N/A	N/A	10	0.623	0.124	24/43(55%)	DTCGenerator 10 2
0	N/A	N/A	10	0.265	0.047	18/43(41%)	DTCGenerator 10
0	N/A	N/A	20	0.268	0.042	19/43(44%)	DTCGenerator 20
5	0.430	0.046	5	0.651	0.095	21/43(48%)	DTCGenerator 5 2, NBCGenerator 5
10	0.391	0.044	10	0.646	0.105	17/43(39%)	DTCGenerator 10 2, NBCGenerator 10
20	0.409	0.045	20	0.661	0.093	22/43(51%)	DTCGenerator 20 2, NBCGenerator 20

Table 1: Table showing the results of our classifiers on the Glass dataset

This dataset was one of the harder ones given to us. The dataset was quite small which made it hard to create a good classifier and the dataset it self was hard to partition into buckets. From the description of the dataset at UCI we could see that not all of the attributes was as important, but this fact is not reflected in our classifiers. From the results above we can see that boosting had very little to do with the result and it seems that the point at which we split the dataset has the most significant importance. The variations between Generators with more classifiers can be explained by the number of times we had to throw away a classifier because it was to bad. This action introduces randomness[6] which affected the results and explains why 20 classifiers for NBC can come up with a different result than 10 classifiers.

2.2 Page-blocks Dataset

This dataset represents the different blocks of the page layout of a document that has been detected by a segmentation process.

To run, use these options with the classifier options wanted.

Listing 2: Page-blocks dataset general options

```
--global 0.2 42 --file page-blocks.txt --filter
no.ntnu.ai.filter.PageBlocksFilter
```

[6]Which is seeded so any attempts to recreate should see the same result

NBC #	Training Error	Standard Deviation	DTC #	Training Error	Standard Deviation	Test Error	Classifier option
1	0.071	N/A	0	N/A	N/A	76/1095(6%)	NBCGenerator 1
0	N/A	N/A	1	0.011	N/A	47/1095(4%)	DTCGenerator 1
5	0.158	0.088	0	N/A	N/A	62/1095(5%)	NBCGenerator 5
10	0.134	0.075	0	N/A	N/A	67/1095(6%)	NBCGenerator 10
20	0.110	0.058	0	N/A	N/A	65/1095(5%)	NBCGenerator 20
0	N/A	N/A	5	0.020	0.010	50/1095(4%)	DTCGenerator 5
0	N/A	N/A	10	0.598	0.263	75/1095(6%)	DTCGenerator 10 1
0	N/A	N/A	10	0.485	0.200	61/1095(5%)	DTCGenerator 10 2
0	N/A	N/A	10	0.021	0.008	47/1095(4%)	DTCGenerator 10
0	N/A	N/A	20	0.021	0.006	50/1095(4%)	DTCGenerator 20
5	0.111	0.023	5	0.533	0.234	66/1095(6%)	DTCGenerator 5 2, NBCGenerator 5
10	0.095	0.007	10	0.596	0.267	56/1095(5%)	DTCGenerator 10 2, NBCGenerator 10
20	0.100	0.004	20	0.678	0.179	53/1095(4%)	DTCGenerator 20 2, NBCGenerator 20

Table 2: Table showing the results of our classifiers on the Page-blocks dataset

This dataset was one of the easier that we were given. As we can see above even the lone classifiers can do very well and boosting has little bearing on the overall result. We can see that the decision tree classifier is the best because it can create fully developed tree which fits very well. The dataset itself also lends itself well to classification as we can see by the single classifiers doing as well as the boosted instances.

2.3 Yeast Dataset

This dataset represents the cellular localization sites of proteins in yeast.

To run, use these options with the classifier options wanted.

Listing 3: Yeast dataset general options

```
--global 0.2 42 --file yeast.txt --filter no.ntnu.ai.
    filter.YeastFilter
```

NBC #	Training Error	Standard Deviation	DTC #	Training Error	Standard Deviation	Test Error	Classifier option
1	0.414	N/A	0	N/A	N/A	132/297(44%)	NBCGenerator 1
0	N/A	N/A	1	0.347	N/A	152/297(51%)	DTCGenerator 1
5	0.585	0.089	0	N/A	N/A	150/297(50%)	NBCGenerator 5
10	0.576	0.064	0	N/A	N/A	148/297(49%)	NBCGenerator 10
20	0.556	0.051	0	N/A	N/A	152/297(51%)	NBCGenerator 20
0	N/A	N/A	5	0.506	0.088	156/297(48%)	DTCGenerator 5
0	N/A	N/A	10	0.796	0.127	187/297(62%)	DTCGenerator 10 1
0	N/A	N/A	10	0.763	0.122	164/297(55%)	DTCGenerator 10 2
0	N/A	N/A	10	0.535	0.076	145/297(48%)	DTCGenerator 10
0	N/A	N/A	20	0.544	0.059	146/297(49%)	DTCGenerator 20
5	0.584	0.088	5	0.796	0.080	141/297(47%)	DTCGenerator 5 2, NBCGenerator 5
10	0.600	0.055	10	0.788	0.126	141/297(47%)	DTCGenerator 10 2, NBCGenerator 10
20	0.598	0.051	20	0.833	0.094	164/297(55%)	DTCGenerator 20 2, NBCGenerator 20

Table 3: Table showing the results of our classifiers on the Yeast dataset

Another hard dataset, the last of the three we had to use. In this dataset we can again see the same trends as in the Glass dataset(2.1). The dataset does not seem to lend it self well to simple classification and we can see that our single classifiers does quite bad. But as with the glass dataset we can see that boosting does not seem to aid us at all. The variance is so large that we can't really say that boosting has an effect and the random element of jiggling the weights makes it hard to give an answer which has some validity in the results.

2.4 Nursery Dataset

This dataset represents the classification of applications for a nursery school.

To run, use these options with the classifier options wanted.

Listing 4: Nursery dataset general options

```
--global 0.2 42 --file nursery.txt --filter no.ntnu.ai.
    filter.NurseryFilter
```

NBC #	Training Error	Standard Deviation	DTC #	Training Error	Standard Deviation	Test Error	Classifier option
1	0.097	N/A	0	N/A	N/A	255/2592(9%)	NBCGenerator 1
0	N/A	N/A	1	0.0	N/A	76/2592(2%)	DTCGenerator 1
5	0.234	0.117	0	N/A	N/A	249/2592(9%)	NBCGenerator 5
10	0.193	0.094	0	N/A	N/A	249/2592(9%)	NBCGenerator 10
20	0.161	0.094	0	N/A	N/A	249/2592(9%)	NBCGenerator 20
0	N/A	N/A	5	0.0	0.0	76/2592(2%)	DTCGenerator 5
0	N/A	N/A	10	0.381	0.066	847/2592(32%)	DTCGenerator 10 1
0	N/A	N/A	10	0.358	0.081	451/2592(17%)	DTCGenerator 10 2
0	N/A	N/A	10	0.0	0.0	76/2592(2%)	DTCGenerator 10
0	N/A	N/A	20	0.0	0.0	76/2592(2%)	DTCGenerator 20
5	0.229	0.075	5	0.380	0.077	291/2592(11%)	DTCGenerator 5 2, NBCGenerator 5
10	0.194	0.075	10	0.349	0.079	313/2592(12%)	DTCGenerator 10 2, NBCGenerator 10
20	0.269	0.037	20	0.347	0.085	338/2592(13%)	DTCGenerator 20 2, NBCGenerator 20

Table 4: Table showing the results of our classifiers on the Nursery dataset

This dataset seems to lend itself well to classification, but again we can see that this does not help our boosting algorithm. We can see that this dataset have a small set of possibilities as the decision tree classifier which can go to maximum depth get the best result. We can see that the naive Bayes classifier does quite well also, but boosting does very little to help to achieve a better result. We can see that boosting does have an effect on the result as we can see the third last does better than just ten decision trees with a depth of two, but it does worse than just ten naive Bayes, which tells us that the boosting does have an effect, but not quite positive. The problem we are seeing with the last tree rows in the table are most likely due to some of the problems that we discuss in the Discussion section(3).

2.5 Pen Digit Dataset

This dataset represents the classification of digits written by different writers.

To run, use these options with the classifier options wanted.

Listing 5: Pen digit dataset general options

```
--global 0.2 42 --file pen-digits.txt --filter no.ntnu.ai
    .filter.PenDigitsFilter
```

NBC #	Training Error	Standard Deviation	DTC #	Training Error	Standard Deviation	Test Error	Classifier option
1	0.119	N/A	0	N/A	N/A	251/2199(11%)	NBCGenerator 1
0	N/A	N/A	1	0.0	N/A	252/2119(11%)	DTCGenerator 1
5	0.314	0.106	0	N/A	N/A	248/2199(11%)	NBCGenerator 5
10	0.329	0.081	0	N/A	N/A	239/2119(10%)	NBCGenerator 10
20	0.330	0.078	0	N/A	N/A	230/2119(10%)	NBCGenerator 20
0	N/A	N/A	5	0.0	0.0	252/2119(11%)	DTCGenerator 5
0	N/A	N/A	10	0.761	0.057	1126/2119(55%)	DTCGenerator 10 1
0	N/A	N/A	10	0.615	0.127	398/2119(18%)	DTCGenerator 10 2
0	N/A	N/A	10	0.0	0.0	252/2119(11%)	DTCGenerator 10
0	N/A	N/A	20	0.0	0.0	252/2119(11%)	DTCGenerator 20
5	0.289	0.086	5	0.664	0.063	172/2119(7%)	DTCGenerator 5 2, NBCGenerator 5
10	0.351	0.049	10	0.612	0.0121	196/2119(8%)	DTCGenerator 10 2, NBCGenerator 10
20	0.347	0.073	20	0.656	0.091	174/2119(7%)	DTCGenerator 20 2, NBCGenerator 20

Table 5: Table showing the results of our classifiers on the Pen-digits dataset

This is one of the only datasets which we can see that boosting is actively helping the solution get better. As we can see the bottom three rows does somewhat better than each of the others individually. This can only come from the effect of boosting as neither the decision tree nor the naive Bayes does better individually than combined. However it does seem that the number of classifiers has little to say, but the combination of decision tree and naive Bayes is what makes this better. If we also look at only five, ten and twenty NBCs we can see that the result does get marginally better, but it's not enough to call it better for our booster.

3 Discussion

One important thing we found when examining the results were that the datasets are very important for the effect the boosting algorithm has. If a dataset has a low number of instances and/or a very heavily skewed proportion of instance-classes boosting, at least using our implementation of Naive Bayesian and Decision Tree Classifiers, is not very effective.

Another important factor for the effect of boosting is the depth of the decision trees the DTC is allowed to construct, if these are deep enough to allow a dataset to be split on all of its attributes it will construct over-fitted trees that only have a marginal improvement from boosting in the cases were the data is ambiguous or unknown.

One of the problems that we encountered when using boosting is that the weight of a classifier would not reflect what it could actually explain. With this we mean that a first classifier might get quite a lot correct and therefor get a high weight, the next classifier will then only care about the instances that the previous classifier got wrong and won't care about the rest. This might mean that the second classifier were able to answer all the instances the first got wrong, but miss classify the rest and thus get a low weight. This would manifest it self when we try to classify the test set, the first classifier would suggest a wrong answer, but because it has a high weight it will win over the second. Even though that test instance might be one of the instances the first had gotten wrong.

This seemed to be a problem for a couple of the other groups and one suggestion was to partition the training set randomly using the weights, possibly with Fitness proportionate selection[7]. When generating a new classifier the classifier would only see a subset of the actual training set, but the weight of the classifier would not be effected by the set of training instances which it did not see. This could mean that the classifiers would only get a weight proportional to the actual explanation strength of that classifier. The intuition behind this goes back to our description above, the first classifier might get a lot correct, but when the second classifier tries to specialize on the instances that the first got wrong it should not be punished for this. Unfortunately we did not have time to try and implement this, but it could possibly solve some of the problems that we experienced during boosting. There are however some possible problems with this approach. For one thing giving each classifier only a subset of the training data might mean that we would get very over fitted classifiers which would go against what AdaBoost tries to accomplish. Another

[7] https://en.wikipedia.org/wiki/Roulette_wheel_selection

problem is the weighting, if we just gave each classifier a subset of the training set, but kept the current way to give classifiers weight almost all classifiers would get a much larger weight which would again give us a problem during testing. This would most likely mean that a highly specialized classifier would get a high weight, but this would also mean that it could "vote" for solutions even though the current instance is outside the classifiers specialization.

One observation that we can make is that for the naive Bayes classifier we can see that the more classifiers we have the less is the standard deviation. This is fully to be expected as we just have more classifiers which generally manages to classify more of the cases and works towards the average.

One of the harder parts about this project was seeing if the booster worked even though the results did not improve dramatically. We have checked several things, but we are quite certain that our implementation is not very wrong. We might have overlooked somethings, but the major part of the project should be intelligent designed. One problem that we have observed is that we have a hard time creating different trees in the DTC. This can easily be observed for some of the datasets where even with five or ten classifiers we got the same average with no deviation. The reason behind this is quite complex, since we do take weights into account when creating the decision trees it should be effected, but when the dataset has little deviation the trees are forced into this configuration. If we grow the trees shallower we can see that the trees are different which does lend some credence to our code. The problem though does mean that boosting is affected because all the trees will "vote" for the same thing overshadowing the other classifiers and skewing the results. This observation also strengthen the theory that the dataset has a large involvement in the outcome of our booster.

4 Conclusion

As we stated what seems to be the biggest problem with our AdaBoost algorithm seems to be the problem that several classifiers "vote" in cases where they should not, but because that classifier is good in the general case it will over turn the "vote" of a classifier that is specialized. The problem could be due to several different things, it might be the way weights are calculated, it might be the way classifiers are given a weight and it might be down to the fact that all classifiers see all training instances even though they should specialize.