Feature bagging for class-based outlier detection

Simon Sudora[1], Peter Aláč[1],

[1] FI MU Brno, Brno, Czech Republic,
{461460, 390904}@mail.muni.cz

Abstract. In this paper, we investigate the impact of using feature bagging method in the context of class-based outlier detection. In particular, we use two methods of feature bagging - Breath-first and Cumulative Sum and apply them with two algorithms – CODB (class-based outlier detection) and RF-OEX (Random Forest - Outlier Explanation). We show, that using feature bagging gives considerably different results and enables improvement of outlier detection.

Keywords: Feature bagging, CODB, class-based outlier detection, RF-OEX, Random Forest-Outlier Explanation

1 Introduction

In this work, we address the problem of class-based outlier detection. Regular outlier detection identifies the instances that are distanced from the other observations. Class-based approach takes the class of an instance into account while identifying outliers. Some instances would not be identified as outliers in the whole set, however they are outliers in the context of their classes. Figure 1. shows an example of such case.

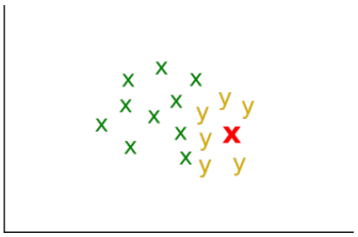

Fig. 1. Example of class-based outlier. Red “x” behaves more like y, however it is labeled as x [2].

The main goal of the project is to apply feature bagging method on the class-based outlier detection algorithm and compare the results. We apply two bagging methods that have been already successfully used with LOF method for global outlier detection, but not for class-based outlier detection [1]. Bagging method firstly creates

n subsets of features from original dataset. These subsets contain from k/2 to k-1 randomly selected features. These features are used as inputs for outlier detection method. After outlier scores are obtained, combining method is applied on the results.

There are two methods for combing results - *Breath-first* and *Cumulative sum.* In the Breath-first method are outlier score vectors firstly sorted in descending (the biggest outliers are always in the first row) order and bound together into matrix. After that it goes through the matrix in the fashion that is shown in the Figure. 2 and it adds examples associated with outlier score into final outlier set. It omits examples, which are already in the final outlier set. This method doesn't compute combined outlier score, it only combines and order identified outliers into final outlier set. The second method used - Cumulative Sum - computes the outlier score for each example simply by summing outlier scores that were computed for each feature subsets.

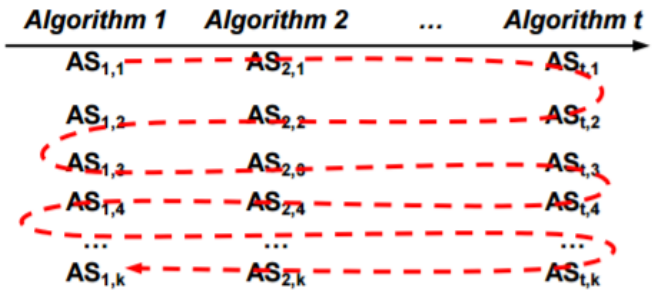

Fig. 2. Illustration of the Breadth-First approach for combining outlier detection scores [4].

We combined these two bagging methods with other two approaches for class based outlier detection - proximity based outlier detection (CODB) and detection using classification (RF-OEX). CODB is combination of distance based and density approach for outlier detection. It computes COF (class outlier factor), which express outlierness of the example in particular class. The second approach we use to identify class outliers was introduced in [3]. It is based on outlier detection using classification, where the instance is identified as an outlier when it is classified into the wrong class. Drawback of this method is great dependence on used classifier. Therefore, we use an ensemble method instead, which combines results of several classifiers. In particular, the method uses random forest ensemble approach.

Random forest uses set of random trees as classifier. Features of random trees are subsets (combination with repetitions) of the original set. They act similarly to regular trees however some level of randomness is introduced in the process of growing the tree – therefore not always the attribute maximizing the information gain (resp. gini index or other measure) is chosen. The proximity of two instances is defined by the number of trees, that classified them into the same class. Based on proximity matrix three metrices are further calculated – distance from the instances of the same class (inversed value of proximity), similarity with other classes and distance within whole set, which is regular class-independent outlier factor. The sum of these three values gives a final outlier factor. The greater it is, more probable is, that the instance is true outlier. We combined these two methods with the technique of bagging to improve

the results of class outlier detection. The implementation procedure and configuration details are described in the following chapter.

2 Implementation

For the two bagging methods, we created our own implementation in the R language. To run outlier detection, we used already existing implementations – for CODB we used implementation in RapidMiner and for the random forest approach we used "Weka-Peka" package implemented by Pekarcikova [3] in Weka. The process workflow is captured in the Figure 3. We tested this approach on two datasets – Zoo and Votes.

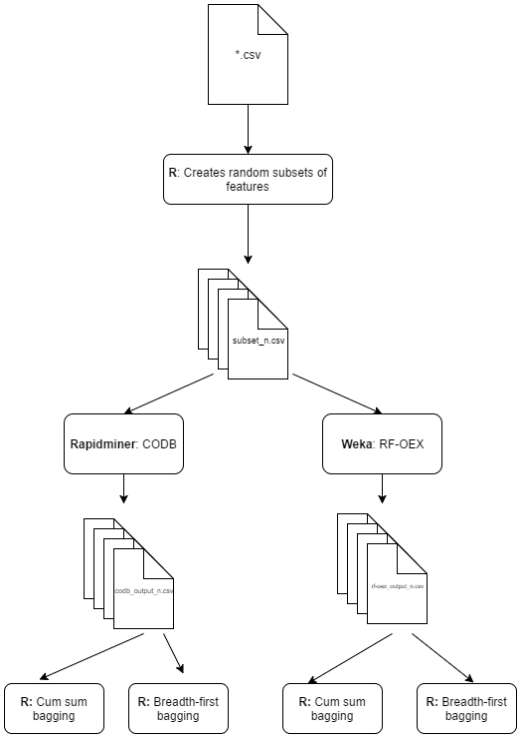

Fig. 3. Process workflow of our "bagged" class-based outlier detection.

2.1 Bagging implementation

Our implementation in R consists of the following scripts:

- **create_features_subsets.R** contains functions for loading datasets and creating random subsets of features as output.

- **algorithms_outputs_integration.R** contains helper functions for loading and handling outputs from Weka and Rapidminer.
- **combine.R** contains functions that implements cumulative sum method and breadth-first method.
- **evaluate.R** contains script for automatized comparison and evaluation of the results.

Cumulative sum: nested for loop that iterates through outlier score matrix and sum final outlier score for each example.

```
for(i in 1:set.size){
  for(j in 1:number.of.subsets){
       result.outliers$score[i] <-
       result.outliers$score[i] +
       outputs.to.combine[[j]][i,"score"]
  }
}
```

Breath-first: nested for loop that iterates through outlier score matrix and adds examples into final set of outliers, but only those that are not already there.

```
for(i in 1:set.size){
 for(j in 1:number.of.subsets){
  if(!(order.vectors[[j]][i]%in% rows.in.result)){
       rows.in.result<-
       c(rows.in.result, order.vectors[[j]][i])
  }
 }
}
```

2.2 RapidMiner CODB configuration

We used default configuration of CODB with 7 neighbors. You can see process flow in the Figure 4.

Fig. 4. Loop that iterates through all the features subsets and run CODB on them.

2.3 Weka-Peka configuration

Weka processes input files only in specific format (.arff), therefore, as the first step, we transformed our datasets into appropriate form. Actual local outlier factor calculation was done using Weka-Peka package within Weka. The configuration used on the *Zoo* dataset is depicted in figure 5. We chose the number of features relatively small (5) to the number of all features (18). On the other hand, the number of the trees was set high, to limit the effect of randomness. The "Number of Outliers for Each Class" was set to the size of the biggest class, so that outlier factor is calculated for each instance. We want to get top n outliers within the whole set not within the classes. The rest of the settings where left to default. As mentioned in [2] using default configuration is sufficient for retrieving reasonable results. Given outputs where extracted and converted into csv form and further processed within R environment. The same procedure was applied on the Votes dataset.

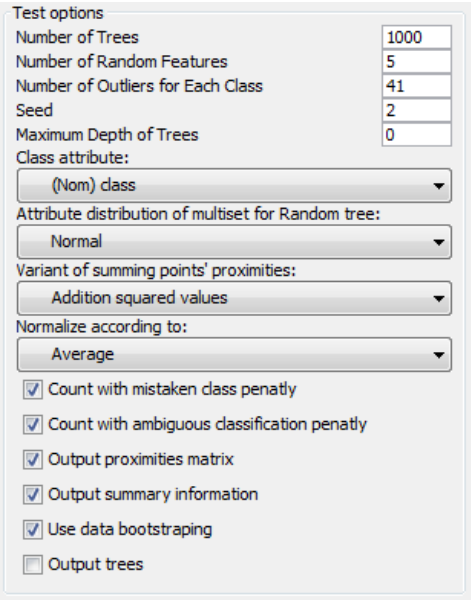

Fig. 5. Weka-Peka configuration for Zoo dataset.

It is also important to mention, that actual RF-OEX [2] enables not only outlier detection, but also interpretation (explanation). However, we only concentrated on detection as we combined the method with bagging method later on.

3 Results

Before we started test implemented bagging methods on Zoo dataset, we have created dendrogram for this dataset to try to identify clear outliers visually.

In dendrogram in the Figure 5 we can see clear clusters of birds, mammals, and fishes. Clusters of mollusca and insect overlay as well as clusters of amphibians and reptiles. We can also identify few candidates for outliers e.g. platypus, seal, porpoise, and dolphin in the class of mammals, seasnake, pitviper and slowworm in the class of reptiles.

We set number of randomly selected features subsets to be 5 for majority of tests. We have also tried and tested different number of subsets in combination with CODB. You can see results in the Table 5.

At first, we ran the two outlier detection algorithms without using feature bagging (captured in tables 1 – 4 always in the second column). We compared our non-bagged RF-OEX results with the referral data given by [2] (first columns in tables 1-4). We found out that outputs are fairly similar. The differences may be caused by different configuration or random aspect, which occurs when random forests are used.

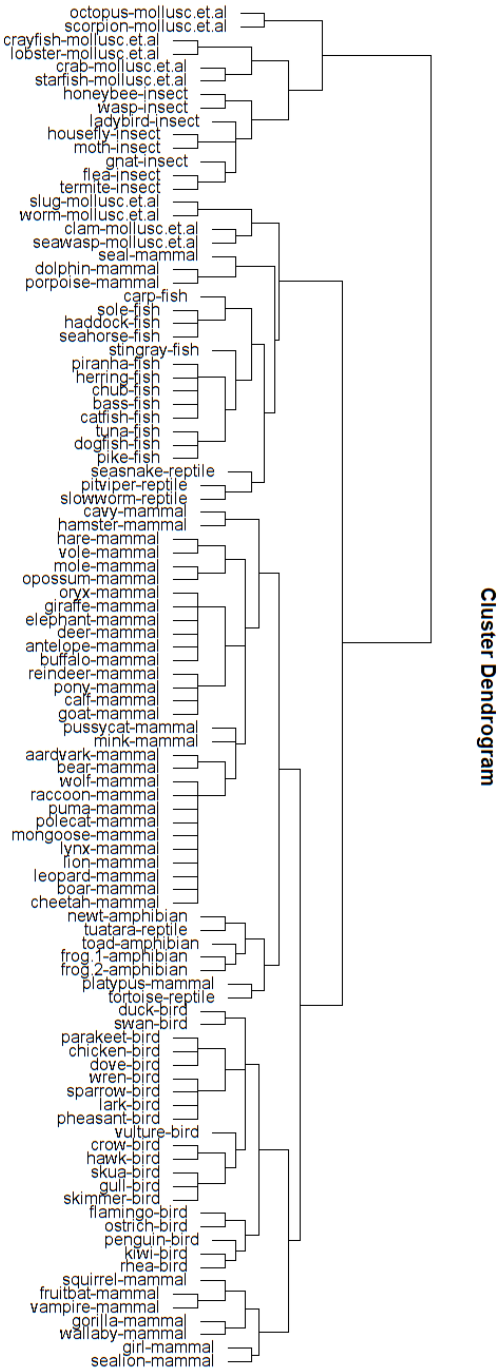

Fig. 6. Dendrogram for Zoo dataset

Table 1. Votes CODB

	votes.referral.outliers	votes.no.bagging.codb	votes.no.bagging.codb.score	votes.codb.cs	votes.codb.cs.score	votes.codb.bf
1	408	421	0.99	421	4.83	421
2	7	281	1.16	281	5.59	281
3	249	177	1.23	177	6.19	177
4	108	100	1.34	100	6.6	100
5	389	370	1.38	370	6.69	370
6	243	172	1.58	86	8.31	172
7	394	86	1.66	172	8.6	86
8	268	55	1.83	55	8.95	336
9	383	349	2.02	349	9.9	55
10	169	2	2.05	428	10.19	355
11	NA	72	2.08	2	10.27	349
12	NA	428	2.09	72	10.28	72
13	NA	276	2.13	29	10.37	2
14	NA	29	2.13	276	10.43	428
15	NA	364	2.14	308	10.52	417

cs – cumulative sum, bf – breath first

Results of bagged methods does not differ much from non-bagged methods. There is 90% match between cumulative sum method and non-bagged method and 80% match between breath-first method and non-bagged method among top 10 outliers. What is interesting that top 5 outliers are completely same in all the three cases.

Table 2. Votes RF-OEX

	votes.referral.outliers	votes.no.bagging.rfoex	votes.no.bagging.rfoex.score	votes.rfoex.cs	votes.rfoex.cs.score	votes.rfoex.bf
1	408	249	73.33	408	50.03	389
2	7	108	38.31	376	43.2	408
3	249	383	35.23	389	42.68	174
4	108	389	23.77	268	40.66	376
5	389	408	21.75	394	37.14	268
6	243	7	18.96	7	33.45	385
7	394	376	18.86	243	31.71	394
8	268	365	8.87	385	31.42	165
9	383	169	8.7	249	29.73	243
10	169	216	8.12	383	28.31	398
11	NA	162	7.61	108	27.84	230
12	NA	152	6.62	72	27.76	7
13	NA	327	6.56	216	27.7	216
14	NA	243	6.47	169	24.87	108
15	NA	394	5.31	152	23.87	383

cs – cumulative sum, bf – breath first

Bagging changed top 10 results of RF-OEX much more than results of CODB. There is 60% match between cumulative sum method and non-bagged method and only 30% match between breath-first method and non-bagged method among top 10 outliers.

Table 3. Zoo CODB

	zoo.referral.outliers	zoo.no.bagging.codb	zoo.no.bagging.codb.score	zoo.codb.cs	zoo.codb.cs.score	zoo.codb.bf
1	platypus	tuatara	2.36	tuatara	8.33	tuatara
2	stingray	slowworm	2.4	seasnake	10.21	seasnake
3	scorpion	dolphin	2.42	tortoise	10.23	tortoise
4	newt	porpoise	2.42	dolphin	10.84	dolphin
5	tortoise	seasnake	2.44	porpoise	10.84	seawasp
6	seasnake	tortoise	2.53	slowworm	13.02	porpoise
7	slug	pitviper	3.48	seal	14.56	crab
8	worm	seal	3.61	pitviper	17.4	seal
9	flea	crayfish	4.2	seawasp	17.65	slowworm
10	termite	lobster	4.2	scorpion	19.53	scorpion
11	NA	starfish	4.24	crab	19.84	crayfish
12	NA	slug	4.28	crayfish	19.99	pitviper
13	NA	worm	4.28	lobster	19.99	lobster
14	NA	clam	4.29	toad	20.95	toad
15	NA	frog.1	4.35	starfish	21.24	slug

cs – cumulative sum, bf – breath first

Bagged methods of CODB for Zoo dataset brought more changes than it was in votes dataset. However, there is still 80% match between cumulative sum method and non-bagged method and 70% match between breath-first. method and non-bagged method among top 10 outliers. Top 4 outliers are same in both bagged methods.

Table 4. Zoo RF-OEX

	zoo.referral.outliers	zoo.no.bagging.rfoex	zoo.no.bagging.rfoex.score	zoo.rfoex.cs	zoo.rfoex.cs.score	zoo.rfoex.bf
1	platypus	platypus	13.13	platypus	36.84	dolphin
2	stingray	dolphin	5.1	porpoise	32	platypus
3	scorpion	scorpion	4.32	dolphin	31.85	porpoise
4	newt	penguin	3.73	scorpion	19.67	scorpion
5	tortoise	ostrich	3.33	penguin	14.58	ostrich
6	seasnake	porpoise	3.3	newt	14.16	haddock
7	slug	seahorse	2.97	piranha	13.99	penguin
8	worm	tortoise	2.97	flea	12.48	piranha
9	flea	termite	2.85	termite	12.21	flea
10	termite	worm	2.61	tortoise	12.17	newt
11	NA	kiwi	2.59	seasnake	11.63	tortoise
12	NA	seasnake	2.55	kiwi	11.23	seahorse
13	NA	ladybird	2.51	ostrich	11.1	seasnake
14	NA	piranha	2.49	ladybird	10.87	herring
15	NA	flea	2.38	worm	10.68	kiwi

cs – cumulative sum, bf – breath first

Bagged results are more consistent with non-bagged results than it was in Votes dataset. There is 70% match between cumulative sum method and non-bagged method and 60% match between breath-first method and non-bagged method among top 10 outliers.

Table 5. Comparison of different numbers of features subsets

	cs.5.subsets	cs.10.subsets	cs.15.subsets	bf.5.subsets	bf.10.subsets	bf.15.subsets
1	tuatara	seasnake	seasnake	tuatara	clam	seasnake
2	seasnake	tortoise	tortoise	seasnake	sealion	clam
3	tortoise	tuatara	tuatara	tortoise	seasnake	platypus
4	dolphin	slowworm	slowworm	dolphin	tortoise	tortoise
5	porpoise	pitviper	pitviper	seawasp	tuatara	dolphin
6	slowworm	dolphin	scorpion	porpoise	slowworm	tuatara
7	seal	porpoise	dolphin	crab	slug	seawasp
8	pitviper	slug	porpoise	seal	worm	scorpion
9	seawasp	worm	slug	slowworm	flea	porpoise
10	scorpion	scorpion	worm	scorpion	dolphin	slowworm
11	crab	seal	newt	crayfish	pitviper	slug
12	crayfish	newt	frog.1	pitviper	termite	pitviper
13	lobster	frog.1	frog.2	lobster	porpoise	kiwi
14	toad	frog.2	seal	toad	scorpion	worm
15	starfish	crayfish	toad	slug	seawasp	crab

cs – cumulative sum, bf – breath first

As we can see increased number of features subsets mostly did not change examples that are in top 10, but it has reordered them significantly.

4 Conclusion

We have implemented a feature bagging method proposed in [4] in R language, which can be now easily used for further experiments. Our implementation allows using two types of feature bagging methods – breadth-first and cumulative sum. Flexible process flow of implemented bagging methods allows to combine them easily with any other outlier detection methods.

We tested impacts of using feature bagging in combination with two methods for class-based outlier detection – CODB and RF-OEX. We found out that usage of

feature bagging has in general greater impact in combination with RF-OEX then with CODB. For RF-OEX methods there was match in the first ten outliers between 60-80% (with one exception on Votes dataset, where it was only 30), and 0-20% perfect match, which represents percentage of items that have same order. For COBD the match was higher – in the range between 70-90%, and with perfect match about 40% in average. Small difference between bagged and non-bagged methods in case of CODB is caused by very similar results for each subset of features. Changing subsets of features had just small impact on the outliers identified by CODB.

The Breath-first method differed from non-bagged results more than cumulative sum in all the test cases. Therefore, the second outcome from tests is that Breath-first bagging method has higher impact on the resulting outliers than cumulative sum.

5 The References

[1] Hewahi, Nabil M., and Motaz K. Saad. "Class outliers mining: Distance-based approach." *International Journal of Intelligent Systems and Technologies* 2 (2007): 5.

[2] Nezvalová L., Popelínský L., Torgo L., Vaculík K. (2015) Class-Based Outlier Detection: Staying Zombies or Awaiting for Resurrection?. In: Fromont E., De Bie T., van Leeuwen M. (eds) Advances in Intelligent Data Analysis XIV. Lecture Notes in Computer Science, vol 9385. Springer, Cham

[3] PEKARČÍKOVÁ, Zuzana. Detekce odlehlých bodů v klasifikovaných datech [online]. Brno, 2013 [cit. 2017-06-06]. Dostupné z: <http://is.muni.cz/th/207719/fi_m/>. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Lubomír Popelínský.

[4] Aleksandar Lazarevic and Vipin Kumar. 2005. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining* (KDD '05). ACM, New York, NY, USA, 157-166. DOI=http://dx.doi.org/10.1145/1081870.10818