WEDNESDAY, OCTOBER 12, 2016

Multinomial Ensemble In H2O

CATEGORIZATION ENSEMBLE IN H2O

An ensemble is the term for using two or more machine learning models together, as a team. There are various types (normally categorized in the literature as boosting, bagging and stacking). In this article I want to look at making teams of high-quality categorizing models. (It handles both multinomial and binomial categorization.)

I will use H2O. Any machine-learning framework that returns probabilities for each of the possible categories can be used, but I will assume a little bit of familiarity with H2O... and therefore I highly recommend my book: Practical Machine Learning with H2O! By the way, I will use R here, though any language supported by H2O can be used.

Getting The Probabilities

Here is the core function (in R). It takes a list of models [1], and runs predict on each of them. The rest of the code is a bit of manipulation to return a 3D array of just the probabilities (`h2o.predict()` returns its chosen answer in the "predict" column, so the `setdiff()` is to get rid of it).

```
getPredictionProbabilities <- function(models, data){
sapply(models, function(m){
  p <- h2o.predict(m, data)
  as.matrix(p[, setdiff(colnames(p), "predict") ])
  }, simplify = "array")
}
```

Using Them

`getPredictionProbabilities()` returns a 3D array:

- First dimension is `nrows(data)`
- Second dimension is the number of possible outcomes (labelled "p1", "p2", ...)
- Third dimension is `length(models)`
- Value is 0.0 to 1.0.

To make the next explanation a bit less abstract, let's assume it is MNIST data (where the goal is to look at raw pixels and guess which of 10 digits it is), and that we have made three models, and that we are trying it on 10,000 test samples. Therefore we have a 10000 x 10 x 3 array. Some example slices of it:

- `predictions[1,,]` is the predictions for the first sample row; returning a 10 x 3 matrix.
- `predictions[1:5, "p3",]` is the probability of each of the first 5 rows being the digit "2". So, five rows and three columns, one per model. (It is "2", because "p1" is "0", "p2" is "1", etc.).
- `predictions[1:5,,1]` are the probabilites of the first model on the first 5 test samples, for each of the 10 categories.

When I just want the predictions of the team, and don't care about the details, I use this wrapper function (explained in the next section):

```
predictTeam <- function(models, data){
  probabilities <- getPredictionProbabilities(models, data)
  apply(
    probabilities, 1,
    function(m) which.max(apply(m, 1, sum))
    )
  }
```

3D Arrays In R

A quick diversion into dealing with multi-dimensional arrays in R. `apply(probabilities, 1, FUN)` will apply FUN to each row in the first dimension. I.e. FUN will be called 10,000 times, and will be given a 10x3 matrix (`m` in the above code). I then use `apply()` on the first dimension of `m`, calling `sum`. `sum()` is called 10

times (once per category) and each time is given a vector of length 3, containing one probability per model.
The last piece of the puzzle is `which.max()`, which tells us which category has the biggest combined confidence from all models. It returns 1 if category 1 (i.e. "0" if MNIST), 2 if category 2 ("1" if MNIST), etc. This gets returned by the function, and therefore gets returned by the outer `apply()`, and therefore is what `predictTeam()` returns.
Aside: Using `sum()` is equivalent to `mean()`, but just very slightly faster. E.g. 30 > 27 > 24, and if you divide through by 3, then 10 > 9 > 8.
Your answer column in your training data is a factor, so to convert that answer into the string it represents, use `levels()`. E.g. if `f` is the factor representing your answer, and `p` is what `predictTeam()`returned, then `levels(f)[p]` will get the string. (With MNIST, there is a short-cut: `p-1` will do it.)

Counting Correctness

A common question to ask of an ensemble is how many test samples every member of the team got right, and how many none of the team got right. I'll look at that in 4-steps. First up is to ask what each models best answer is.[3]

```
predictByModel <- apply(probabilities, 1,
 function(sample) apply(sample, 2, which.max)
 )
```

Assuming the earlier MNIST example, this gives a 3 x 10000 int matrix (where the values are 1 to 10). Next, we need a vector of the correct answers, and they must be the category index, not the actual value. (In the case of MNIST data it is simply adding 1, so "0" becomes 1, "1" becomes 2, etc.)

```
eachModelsCorrectness <- apply(predictByModel, 1,
 function(modelPredictions) modelPredictions == correctAnswersTest
 )
```

This returns a 10000 x 3 logical (boolean) matrix. It tells us if each model got each answer correct. The next step returns a 10000 element vector, which will have 0 if they all got it wrong, 1 if only one member of the team got the correct answer, 2 if two of them, and 3 if all three members of the team got it right.

```
cmc <- apply(eachModelsCorrectness, 1, sum)
```

(`cmc` for count of model correctness) `table(cmc)` will give you those four numbers, E.g. you might see:

```
 0    1    2    3
67   59  105 9769
```

Indicating that there were 67 that none of them got right, 59 samples that only one of the three models managed, 105 that just one model got wrong, and 9769 that all the models could do.
I sometimes find `cumsum(table(cmc))` to be more useful:

```
 0     1     2     3
67   126   231 10000
```

Either way, if the first number is high, you need stronger model(s). Better team members. If it is low, but the ensemble is performing poorly (i.e. no better than the best individual model), then you need more variety in the models.

Further Ideas

By slicing and dicing the 3D probability array, you can do different things. You might want to poke into what those difficult 67 samples were, that none of your models can get. Another idea, which I will deal in a future blog post, is how you can use the probability to indicate those with low confidence that need more investigation work. (You can use this with an ensemble, or just with a single model.)

Performance

In my experience, in general, this kind of team will always give better performance than any individual model in the team. It gives the biggest jump in strength, when:

- The models are very distinct.
- The models are fairly close in strength.
- Just a few models. Making 4 or 5 models, evaluating them on the valid data, and using the strongest 3 for the ensemble, works reasonably well. However if you managing to make *near-strength* and *very distinct* models, the more the merrier.

For instance, three deep learning models, all built with the same parameters, each getting about 2% error individually, might manage 1.8% error as a team. But two deep learning models, one random forest, one

GBM, all getting 2% error, might give 1.5% or even lower, when used together. Conversely, two very similar deep learning models that have 2% error combined with a GBM with 3% error and a random forest getting 4% error might do poorly, possibly even worse than your single best model.

Summary

This approach to making an ensemble for categorization problems is straightforward and usually very effective. Because it works with models of different types it at first glance seems closer to stacking, but really it is more like the bagging of random forest (except using probabilities, instead of mode[4] as in random forest).

Footnotes

[1]: All the models in `models` must all be modelling the same thing, i.e. they must each have been trained on data with the exact same columns.[2]
Normally each model will have been trained on the exact same data, but the precise phrasing is deliberate: interesting ensembles can be built with models trained on different subsets of your data. (By the way, subsets and ensemble are the two fundamental ideas behind the random forest algorithm.)
[2]: You can take this even further and only require they are all trained with the same response variable, not even the same columns. However, you'll need to adapt `getPredictionProbabilities()` in that case, because each call to `h2o.predict()` will then need to be given a different `data` object.
[3]: I do it this way because I already had `probabilities` made. But if it was all that you were interested in, this was the long way to get it. The direct way was to do the exact opposite of `getPredictionProbabilities()`: keep the "predict" column, and get rid of the other columns!
[4]: I did experiment with simple one vote per model (aka mode), instead of summing probabilities, and generally got worse results. But it is worth bearing that kind of ensemble in mind. If you do more rigorous experiments comparing the two methods, I'd be very interested to hear what kind of problems, if any, that voting (mode) is superior on.

POSTED BY UNKNOWN AT 9:12 AM

LABELS: ARTIFICIAL INTELLIGENCE, ENSEMBLES, H2O, LEARNING ALGORITHMS, MACHINE LEARNING, MNIST, R

NO COMMENTS:

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

ABOUT THIS BLOG

This is a blog of tutorials, code samples, workarounds for problems, etc, etc. that I meet in my daily coding. It is as much for me, so I do not forget, as for you!

A range of languages and topics; use the Labels to narrow things down.

I will try to edit the posts to keep them current and useful, so you don't need to trawl all the comments as well as read the article. (This means if I use a suggestion from a comment I'll delete that comment; but I will credit you.)

ABOUT ME

UNKNOWN

VIEW MY COMPLETE PROFILE

BLOG ARCHIVE

► 2008 (14)