

The Shape of Data

Exploring the geometry behind machine learning, data mining, etc.

Multi-class classification

Posted on [June 4, 2013](#)

If you paid really close attention to my last few posts, you might have noticed that I've been cheating slightly. (But if you didn't notice, don't worry – it was a subtle cheat.) When I introduced the problem of classification, the main example was choosing a restaurant to recommend to someone. So, most likely we were selecting a restaurant from a fairly long list – half a dozen to perhaps a few hundred depending on where you live. This is called *multi-class classification* (not to be confused with [multi-label classification](#)). However, when I described [SVM](#) and [logistic regression](#) algorithms, I only ever looked at two classes, which would mean selecting from between two different restaurants. This is called *binary classification*. There's a good reason why I did this – binary classification algorithms are much easier to come by than multi-class algorithms. In fact, there are relatively few algorithms specifically for multi-class classification. As it turns out, it is possible to apply a binary classifier repeatedly and combine the results to get a multi-class classification, and this is the much more common procedure. In this post, I will discuss a few different ways of doing this.

The process for turning a binary classifier into a multi-class classifier is kind of like a game of twenty questions. However, rather than being allowed to ask any yes/no question, you can only ask questions of the form “Is this data point more likely to be in class A or in class B?” This is a problem because if the data point isn't in either of class A or class B, then you will still get an answer – it just may be meaningless.

So for example, one scheme you might try is to run your binary classifier for every possible pair of classes and treat it as an election – whichever class gets the most votes wins. If you have three classes of data (red, blue and green) you would train one binary classifier for each pair of classes: 1) red vs. blue, 2) blue vs. green and 3) red vs. green. If you use a linear classifier like SVM or logistic regression, then this will determine three different lines/planes/hyperplanes (depending on the dimension of the data). We will call these lines/planes/hyperplanes the *decision boundaries*.

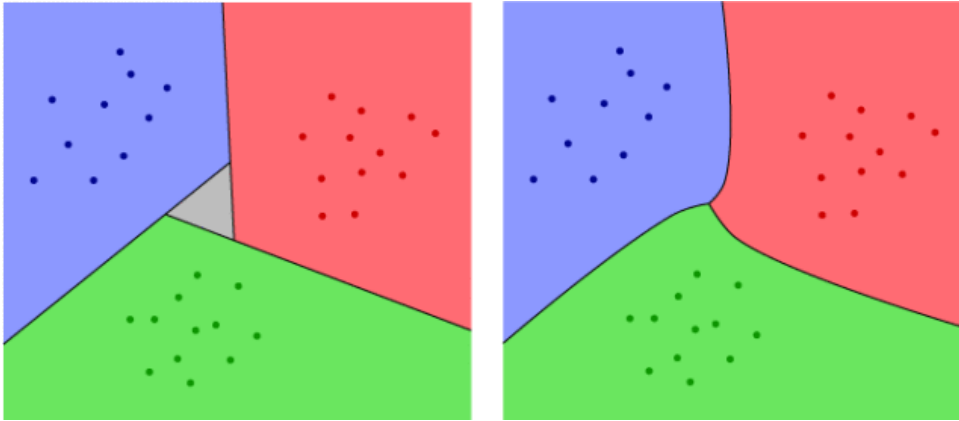
When new data point comes in, you check which side of each decision boundary it is on and see which class is most popular: If a new data point is on the green side of both the red vs. green and the blue vs. green hyperplanes then you classify it as green. In this case, the red vs. blue classifier is meaningless, but that's OK because we ignore that one anyway. If we use a non-linear classifier or run our linear classifier through a kernel then the decision boundaries will be curved shapes rather than flat lines/planes/hyperplanes, but we can combine them the same way.

This scheme is called All vs. All (AVA). The problem is that there will usually be regions in the vector space of all data points in which the three different classifiers will produce a three-way tie. A good way to think about this is to visualize the two-dimensional case with three classes (such as our red, green and blue). An example is shown on

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept

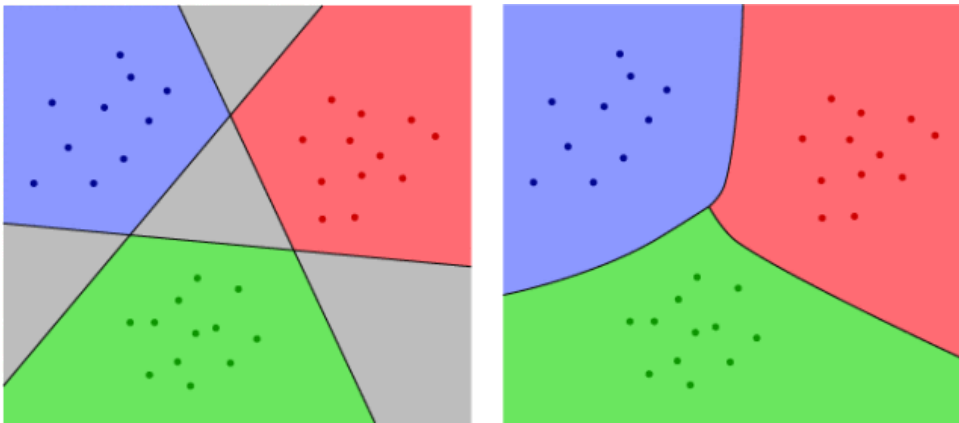
But because the three instances of the binary classifier are independent, and all are affected by noise, there is bound to be some grey region.



One way to get rid of the grey region is to use a classifier that includes a measure of confidence. For example, logistic regression doesn't just give us a decision boundary that separates the two classes – it gives us a probability density function, which was represented in my earlier post as a slow transition from green to blue. The shade of green or blue at any given point can be thought of as indicating how confident the algorithm is that a given point is in one class or the other. So, rather than just adding the number of votes for each class, we can add up the confidence values. Since these can be any values, rather than just ones and zeros, it will be much less likely to get a tie. The resulting algorithm will give us an answer at every point except for the points in the three curves where we switch from one color to another. In our red/green/blue example, this will look like the right side of the Figure above. (The pictures are hand drawn, so they are not 100% accurate, but they should give you the basic idea.)

Another popular scheme is to set up for each class a classifier that compares it to all the other classes at once. For our red, green and blue classes, we would train one classifier for each of: 1) red vs. (green or blue), 2) green vs. (red or blue) and 3) blue vs. (red or green). In other words, the first classifier would predict whether a given point is more likely to be in the red class, or in one of the blue or green classes, and similarly for the second and third classifier. We then choose a class for any new data point based on whichever class wins its own election. This scheme is called One vs. All (OVA).

With this scheme, there are two possible things that can go wrong: For a new data point, there may be two or more classes that win their own election (i.e. are determined to be more likely than the rest). Or, there may be no data points for which an individual class wins an election. You can get a rough idea of this from the left side of the Figure below. The grey regions on the outside are the areas where two different classes win their own elections. The grey triangle in the middle is where no class wins. This problem can again be minimized by using a classifier that includes a confidence value, as indicated on the right of the figure.



From the picture, it looks like there isn't a big difference between the OVA and AVA results, particularly when a confidence value is included. All the methods certainly agree in the regions closest to the different classes of data, which is what we would expect, since there is an obvious answer in these areas. They mostly disagree in the regions between the data sets, where we wouldn't expect as many new data points to appear. In particular, if the data is determined by well separated probability distributions then the different methods will usually give us the same answer, since most of the new data points will fall near the center of the appropriate class and away from the decision boundaries. However, if the models defined by the individual classifiers are inaccurate or if the different classes of data points are not well separated, AVA and OVA may give you very different results, and which one works better will depend on the situation.

A third scheme for combining binary classifiers uses something more like the usual strategy for a twenty questions game. The idea is to start with very general questions and then slowly narrow down the results. For example, if someone is thinking of a number between 1 and 100 and you want to figure out what it is with twenty yes/no questions, your first question shouldn't be "Is the number 1?". Instead, you would ask whether or not the number was greater than 50. If the answer is no, then you would ask whether or not the number is greater than 25, and so on. In other words, you try to cut the possibilities in half with each question.

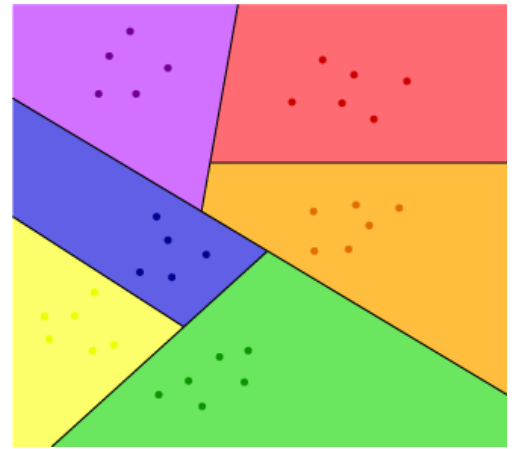
We can do the same thing with binary classifiers: If we have a large number of classes, we can divide them into two sets of classes, say A and B. Then we can divide the classes in A into two smaller sets of classes, divide B into two smaller sets of classes and so on. To run our multi-class classification, we would first train a binary classifier to determine whether a new data point is in some class in A, or in some class in B. (So, this step wouldn't determine *which* class in A the data point is in, just whether or not it's on one of them.) We then train a second binary classifier to determine which of the two subsets of A a point is in, and a third classifier to determine which of the subsets of B a point is in. We continue all the way down, until we get to classifiers that distinguish individual classes. This is called *hierarchical classification* because the different steps in the scheme form a sort of hierarchy from the first question (the CEO) to the second level questions (the vice-presidents) and down to the final questions (the mailroom clerks) that distinguish individual classes. The resulting structure, if you do this with a linear classifier, is indicated in the figure below. (Note that this is closely related to a method called decision trees, which I'll discuss in a later post.)

In the Figure, set A consists of the three classes (purple, red and orange) in the upper right and set B consists of

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept

This last scheme has the advantage that even without using a confidence indicator from the binary classifier, it always gives you a well defined answer for every new data point (as long as it isn't right in one of the decision boundaries.) In other words, the hierarchical classification scheme does not produce any grey regions. The tricky thing is that how you choose to construct the hierarchy can have a big impact on how effective the final classifier is. In particular, you want to make sure that each of your initial sets A or B combines classes that are naturally close together in the space containing the data points. Otherwise, there may not be a hyperplane that separates A from B. Of course, we can always use a kernel to separate them, but as a rule of thumb it's better to arrange things so that you can use the simplest possible kernel.



So, repeating a common theme on this blog, there is no silver bullet (or [free lunch](#)) when it comes to choosing a scheme for combining binary classifiers. The best way to make the right decision is to use a combination of experience with the type of data your working on and an understanding of the structure of the data, based on experimentation, visualization and other types of exploratory techniques. Otherwise, since you can't actually see the data and the model in the same way that I've drawn the above examples, it can be very difficult to tell when a model or classification scheme is really effective or just deceptive garbage.

Advertisements

Share this:



One blogger likes this.

Related

[Nearest Neighbors Classification](#)
In "Classification"

[Decision Trees](#)
In "Classification"

[Neural Networks 1: The neuron](#)
In "Classification"

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept

Pingback: [Decision Trees](#) | [The Shape of Data](#)

Pingback: [Random forests](#) | [The Shape of Data](#)

Pingback: [The shape of data](#) | [spider's space](#)



[triangleinequality](#) says:

March 5, 2014 at 6:57 am

Hi I am enjoying reading your blog. In the tradeoff between one vs all and all vs all strategies, perhaps the most important is computation time. OVA requires N models, where N is the number of classes, whereas AVA requires $N(N-1)/2$.

The AVA models use less data however, and so will be cheaper to train. So it depends on how your model scales on the number of data points which will be cheaper to use.

[Reply](#)



[Jesse Johnson](#) says:

March 5, 2014 at 9:40 am

That's a good point. I had been thinking about this in terms of a relatively small number of classes, so the computational complexity relative to N wouldn't be an issue. But I think you're right that in many situations, you'll want the number of classes being large enough that AVA is computationally unfeasible.

[Reply](#)



[Praveen Kumar](#) says:

October 6, 2015 at 12:02 am

Small typo: "Then in the final two steps, we separate red from orange and yellow from green." should be "Then in the final two steps, we separate red from orange and yellow from BLUE." Great post otherwise 😊

[Reply](#)



[Jesse Johnson](#) says:

October 8, 2015 at 8:26 pm

Fixed. Thanks!

[Reply](#)



[Milan Rathod](#) says:

December 30, 2015 at 1:54 am

Hi Jesse,

I am trying to do multi class classification using FLANN + LMS . but I donot know how to form cost function, decision boundry and performance criteria to compare results with other architectures

[Reply](#)

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept

Pingback: [MACHINE LEARNING – REMOTE SENSING – Blog Features Imagery.](#)

Pingback: [MACHINE LEARNING – REMOTE SENSING – Home](#)

The Shape of Data

Create a free website or blog at WordPress.com.

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept