Draft Version

MACHINE LEARNING YEARNING

Technical Strategy for AI Engineers, In the Era of Deep Learning

ANDREW NG

deeplearning.ai

Machine Learning Yearning is a
deeplearning.ai project.

15 Evaluating multiple ideas in parallel during error analysis

Your team has several ideas for improving the cat detector:

- Fix the problem of your algorithm recognizing *dogs* as cats.
- Fix the problem of your algorithm recognizing *great cats* (lions, panthers, etc.) as house cats (pets).
- Improve the system's performance on *blurry* images.
- ...

You can efficiently evaluate all of these ideas in parallel. I usually create a spreadsheet and fill it out while looking through ~100 misclassified dev set images. I also jot down comments that might help me remember specific examples. To illustrate this process, let's look at a spreadsheet you might produce with a small dev set of four examples:

Image	Dog	Great cat	Blurry	Comments
1	✔			Unusual pitbull color
2			✔	
3		✔	✔	Lion; picture taken at zoo on rainy day
4		✔		Panther behind tree
% of total	25%	50%	50%	

Image #3 above has both the Great Cat and the Blurry columns checked. Furthermore, because it is possible for one example to be associated with multiple categories, the percentages at the bottom may not add up to 100%.

Although you may first formulate the categories (Dog, Great cat, Blurry) then categorize the examples by hand, in practice, once you start looking through examples, you will probably be inspired to propose new error categories. For example, say you go through a dozen images and realize a lot of mistakes occur with Instagram-filtered pictures. You can go back and add a new "Instagram" column to the spreadsheet. Manually looking at examples that the algorithm misclassified and asking how/whether you as a human could have labeled the

picture correctly will often inspire you to come up with new categories of errors and solutions.

The most helpful error categories will be ones that you have an idea for improving. For example, the Instagram category will be most helpful to add if you have an idea to "undo" Instagram filters and recover the original image. But you don't have to restrict yourself only to error categories you know how to improve; the goal of this process is to build your intuition about the most promising areas to focus on.

Error analysis is an iterative process. Don't worry if you start off with no categories in mind. After looking at a couple of images, you might come up with a few ideas for error categories. After manually categorizing some images, you might think of new categories and re-examine the images in light of the new categories, and so on.

Suppose you finish carrying out error analysis on 100 misclassified dev set examples and get the following:

Image	Dog	Great cat	Blurry	Comments
1	✔			Usual pitbull color
2			✔	
3		✔	✔	Lion; picture taken at zoo on rainy day
4		✔		Panther behind tree
...	...	...	...	...
% of total	8%	43%	61%	

You now know that working on a project to address the Dog mistakes can eliminate 8% of the errors at most. Working on Great Cat or Blurry image errors could help eliminate more errors. Therefore, you might pick one of the two latter categories to focus on. If your team has enough people to pursue multiple directions in parallel, you can also ask some engineers to work on Great Cats and others to work on Blurry images.

Error analysis does not produce a rigid mathematical formula that tells you what the highest priority task should be. You also have to take into account how much progress you expect to make on different categories and the amount of work needed to tackle each one.

16 Cleaning up mislabeled dev and test set examples

During error analysis, you might notice that some examples in your dev set are mislabeled. When I say "mislabeled" here, I mean that the pictures were already mislabeled by a human labeler even before the algorithm encountered it. I.e., the class label in an example *(x,y)* has an incorrect value for *y*. For example, perhaps some pictures that are not cats are mislabeled as containing a cat, and vice versa. If you suspect the fraction of mislabeled images is significant, add a category to keep track of the fraction of examples mislabeled:

Image	Dog	Great cat	Blurry	Mislabeled	Comments
...					
98				✔	Labeler missed cat in background
99		✔			
100				✔	Drawing of a cat; not a real cat.
% of total	8%	43%	61%	6%	

Should you correct the labels in your dev set? Remember that the goal of the dev set is to help you quickly evaluate algorithms so that you can tell if Algorithm A or B is better. If the fraction of the dev set that is mislabeled impedes your ability to make these judgments, then it is worth spending time to fix the mislabeled dev set labels.

For example, suppose your classifier's performance is:

- Overall accuracy on dev set..... 90% (10% overall error.)
- Errors due to mislabeled examples..... 0.6% (6% of dev set errors.)
- Errors due to other causes..... 9.4% (94% of dev set errors)

Here, the 0.6% inaccuracy due to mislabeling might not be significant enough relative to the 9.4% of errors you could be improving. There is no harm in manually fixing the mislabeled images in the dev set, but it is not crucial to do so: It might be fine not knowing whether your system has 10% or 9.4% overall error.

Suppose you keep improving the cat classifier and reach the following performance:

- Overall accuracy on dev set..... 98.0% (2.0% overall error.)
- Errors due to mislabeled examples..... 0.6%. (30% of dev set errors.)
- Errors due to other causes..... 1.4% (70% of dev set errors)

30% of your errors are due to the mislabeled dev set images, adding significant error to your estimates of accuracy. It is now worthwhile to improve the quality of the labels in the dev set. Tackling the mislabeled examples will help you figure out if a classifier's error is closer to 1.4% or 2%—a significant relative difference.

It is not uncommon to start off tolerating some mislabeled dev/test set examples, only later to change your mind as your system improves so that the fraction of mislabeled examples grows relative to the total set of errors.

The last chapter explained how you can improve error categories such as Dog, Great Cat and Blurry through algorithmic improvements. You have learned in this chapter that you can work on the Mislabeled category as well—through improving the data's labels.

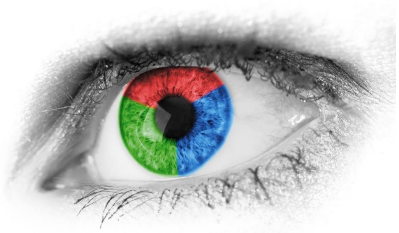
Whatever process you apply to fixing dev set labels, remember to apply it to the test set labels too so that your dev and test sets continue to be drawn from the same distribution. Fixing your dev and test sets together would prevent the problem we discussed in Chapter 6, where your team optimizes for dev set performance only to realize later that they are being judged on a different criterion based on a different test set.

If you decide to improve the label quality, consider double-checking both the labels of examples that your system misclassified as well as labels of examples it correctly classified. It is possible that both the original label and your learning algorithm were wrong on an example. If you fix only the labels of examples that your system had misclassified, you might introduce bias into your evaluation. If you have 1,000 dev set examples, and if your classifier has 98.0% accuracy, it is easier to examine the 20 examples it misclassified than to examine all 980 examples classified correctly. Because it is easier in practice to check only the misclassified examples, bias does creep into some dev sets. This bias is acceptable if you are interested only in developing a product or application, but it would be a problem if you plan to use the result in an academic research paper or need a completely unbiased measure of test set accuracy.

17 If you have a large dev set, split it into two subsets, only one of which you look at

Suppose you have a large dev set of 5,000 examples in which you have a 20% error rate. Thus, your algorithm is misclassifying ~1,000 dev images. It takes a long time to manually examine 1,000 images, so we might decide not to use all of them in the error analysis.

In this case, I would explicitly split the dev set into two subsets, one of which you look at, and one of which you don't. You will more rapidly overfit the portion that you are manually looking at. You can use the portion you are not manually looking at to tune parameters.



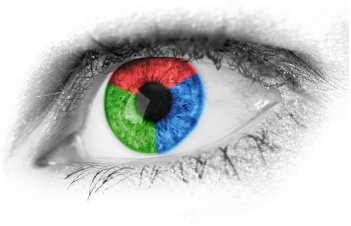
Let's continue our example above, in which the algorithm is misclassifying 1,000 out of 5,000 dev set examples. Suppose we want to manually examine about 100 errors for error analysis (10% of the errors). You should randomly select 10% of the dev set and place that into what we'll call an **Eyeball dev set** to remind ourselves that we are looking at it with our eyes. (For a project on speech recognition, in which you would be listening to audio clips, perhaps you would call this set an Ear dev set instead). The Eyeball dev set therefore has 500 examples, of which we would expect our algorithm to misclassify about 100.

The second subset of the dev set, called the **Blackbox dev set**, will have the remaining 4500 examples. You can use the Blackbox dev set to evaluate classifiers automatically by measuring their error rates. You can also use it to select among algorithms or tune hyperparameters. However, you should avoid looking at it with your eyes. We use the term "Blackbox" because we will only use this subset of the data to obtain "Blackbox" evaluations of classifiers.

Why do we explicitly separate the dev set into Eyeball and Blackbox dev sets? Since you will gain intuition about the examples in the Eyeball dev set, you will start to overfit the Eyeball dev set faster. If you see the performance on the Eyeball dev set improving much more rapidly than the performance on the Blackbox dev set, you have overfit the Eyeball dev set. In this case, you might need to discard it and find a new Eyeball dev set by moving more examples from the Blackbox dev set into the Eyeball dev set or by acquiring new labeled data.

Explicitly splitting your dev set into Eyeball and Blackbox dev sets allows you to tell when your manual error analysis process is causing you to overfit the Eyeball portion of your data.

18 How big should the Eyeball and Blackbox dev sets be?



Your Eyeball dev set should be large enough to give you a sense of your algorithm's major error categories. If you are working on a task that humans do well (such as recognizing cats in images), here are some rough guidelines:

- An eyeball dev set in which your classifier makes 10 mistakes would be considered very small. With just 10 errors, it's hard to accurately estimate the impact of different error categories. But if you have very little data and cannot afford to put more into the Eyeball dev set, it's better than nothing and will help with project prioritization.
- If your classifier makes ~20 mistakes on eyeball dev examples, you would start to get a rough sense of the major error sources.
- With ~50 mistakes, you would get a good sense of the major error sources.
- With ~100 mistakes, you would get a very good sense of the major sources of errors. I've seen people manually analyze even more errors—sometimes as many as 500. There is no harm in this as long as you have enough data.

Say your classifier has a 5% error rate. To make sure you have ~100 mislabeled examples in the Eyeball dev set, the Eyeball dev set would have to have about 2,000 examples (since 0.05*2,000 = 100). The lower your classifier's error rate, the larger your Eyeball dev set needs to be in order to get a large enough set of errors to analyze.

If you are working on a task that even humans cannot do well, then the exercise of examining an Eyeball dev set will not be as helpful because it is harder to figure out why the algorithm didn't classify an example correctly. In this case, you might omit having an Eyeball dev set. We discuss guidelines for such problems in a later chapter.

How about the Blackbox dev set? We previously said that dev sets of around 1,000-10,000 examples are common. To refine that statement, a Blackbox dev set of 1,000-10,000 examples will often give you enough data to tune hyperparameters and select among models, though there is little harm in having even more data. A Blackbox dev set of 100 would be small but still useful.

If you have a small dev set, then you might not have enough data to split into Eyeball and Blackbox dev sets that are both large enough to serve their purposes. Instead, your entire dev set might have to be used as the Eyeball dev set—i.e., you would manually examine all the dev set data.

Between the Eyeball and Blackbox dev sets, I consider the Eyeball dev set more important (assuming that you are working on a problem that humans can solve well and that examining the examples helps you gain insight). If you only have an Eyeball dev set, you can perform error analyses, model selection and hyperparameter tuning all on that set. The downside of having only an Eyeball dev set is that the risk of overfitting the dev set is greater.

If you have plentiful access to data, then the size of the Eyeball dev set would be determined mainly by how many examples you have time to manually analyze. For example, I've rarely seen anyone manually analyze more than 1,000 errors.

19 Takeaways: Basic error analysis

- When you start a new project, especially if it is in an area in which you are not an expert, it is hard to correctly guess the most promising directions.
- So don't start off trying to design and build the perfect system. Instead build and train a basic system as quickly as possible—perhaps in a few days. Then use error analysis to help you identify the most promising directions and iteratively improve your algorithm from there.
- Carry out error analysis by manually examining ~100 dev set examples the algorithm misclassifies and counting the major categories of errors. Use this information to prioritize what types of errors to work on fixing.
- Consider splitting the dev set into an Eyeball dev set, which you will manually examine, and a Blackbox dev set, which you will not manually examine. If performance on the Eyeball dev set is much better than the Blackbox dev set, you have overfit the Eyeball dev set and should consider acquiring more data for it.
- The Eyeball dev set should be big enough so that your algorithm misclassifies enough examples for you to analyze. A Blackbox dev set of 1,000-10,000 examples is sufficient for many applications.
- If your dev set is not big enough to split this way, just use an Eyeball dev set for manual error analysis, model selection, and hyperparameter tuning.