Error Analysis

The idea of prioritizing what to work on is perhaps the most important skill programmers typically need to develop and it's so easy to have many ideas we want to work on. In this lesson, we're going to talk about the concept of error analysis. Which will hopefully give us a way to more systematically make some of these decisions.

Recommended approach

If we're starting work on a machine learning problem, or building a machine learning application. It's often considered very good practice to start, not by building a very complicated system with lots of complex features, instead of that we can:

- **Start with a simple algorithm that we can implement quickly. Implement it and test it on our cross-validation data.**
- **Plot learning curves to decide of more data, more features, etc. are likely to help** (the reason that this is a good approach is often, when we're just starting out on a learning problem, there's really no way to tell in advance, whether we need more complex features, or whether we need more data, or something else, and it's just very hard to tell in advance).
- **Error analysis: Manually examine the examples (in cross validation set) that our algorithm made errors on. See if our spot any systematic trend in what type of examples it is making errors on** (this is the process that will inspire us to design new features, or they'll tell us what are the current things or current shortcomings of the system, and give us the inspiration that we need to come up with improvements to it).

Let's say we've built a spam classifier and we have 500 examples in your cross validation set, and let's say in this example that the algorithm has a very high error rate.

- $m_{CV} = 500$ examples in cross validation set

Algorithm misclassifies 100 emails.

- Manually examine the 100 errors, and categorize them based on:
  - (i) what type of email it is (pharma, replica/fake, steal password, ...). See which type is most common - focus our work on those ones.
  - (ii) What cues (features) we think would have helped the algorithm them correctly (deliberate misspelling, unusual email routing, unusual puntuation), may fine some "spammer technique" is causing a lot of our misses.

The importance of numerical evaluation

If we're developing an algorithm, it's really good to have some performance calculation which gives a single real number to tell us how well its doing. Say were deciding if we should treat a set of similar words as the same word:

- Say were deciding if we should treat a set of similar words as the same word. This is done by stemming in NLP (e.g. "Porter stemmer" looks at the etymological stem of a word)
  - This may make our algorithm better or worse
  - Also worth consider weighting error (false positive vs. false negative)
    - e.g. is a false positive really bad, or is it worth have a few of one to improve performance a lot
- Can use numerical evaluation to compare the changes
  - See if a change improves an algorithm or not
- A single real number may be hard/complicated to compute

    - But makes it much easier to evaluate how changes impact our algorithm
- We should do error analysis on the cross validation set instead of the test set

Error analysis may not be helpful for deciding if this is likely to improve performance. Only solution is to try it and see if it works.

Video Question: Why is the recommended approach to perform error analysis using the cross validation data used to compute $J_{CV}(\theta)$ rather than the test data used to compute $J_{test}(\theta)$?

- The cross validation data set is usually large.
- This process will give a lower error on the test set.

> If we develop new features by examining the test set, then we may end up choosing features that work well specifically for the test set, so $J_{test}(\theta)$ is no longer a good estimate of how well we generalize to new examples.

- Doing so is less likely to lead to choosing an excessive number of features.

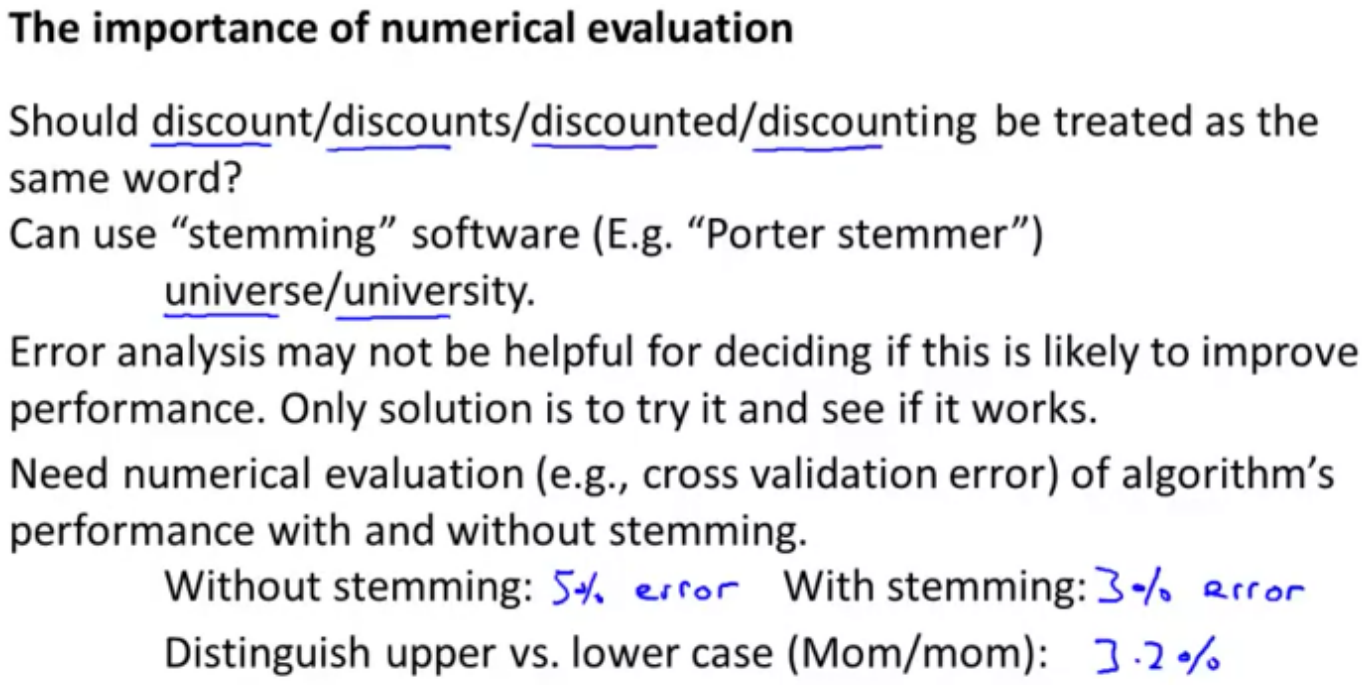
Summary

The recommended approach to solving machine learning problems is to:

- Start with a simple algorithm, implement it quickly, and test it early on your cross validation data.
- Plot learning curves to decide if more data, more features, etc. are likely to help.
- Manually examine the errors on examples in the cross validation set and try to spot a trend where most of the errors were made.

For example, assume that we have 500 emails and our algorithm misclassifies a 100 of them. We could manually analyze the 100 emails and categorize them based on what type of emails they are. We could then try to come up with new cues and features that would help us classify these 100 emails correctly.

Hence, if most of our misclassified emails are those which try to steal passwords, then we could find some features that are particular to those emails and add them to our model. We could also see how classifying each word according to its root changes our error rate:



It is very important to get error results as a single, numerical value. Otherwise it is difficult to assess your algorithm's performance. For example if we use stemming, which is the process of treating the same word with different forms (fail/failing/failed) as one word (fail), and get a 3% error rate instead of 5%, then we should definitely add it to our model.

However, if we try to distinguish between upper case and lower case letters and end up getting a 3.2% error rate instead of 3%, then we should avoid using this new feature. Hence, we should try new things, get a numerical value for our error rate, and based on our result decide whether we want to keep the new feature or not.