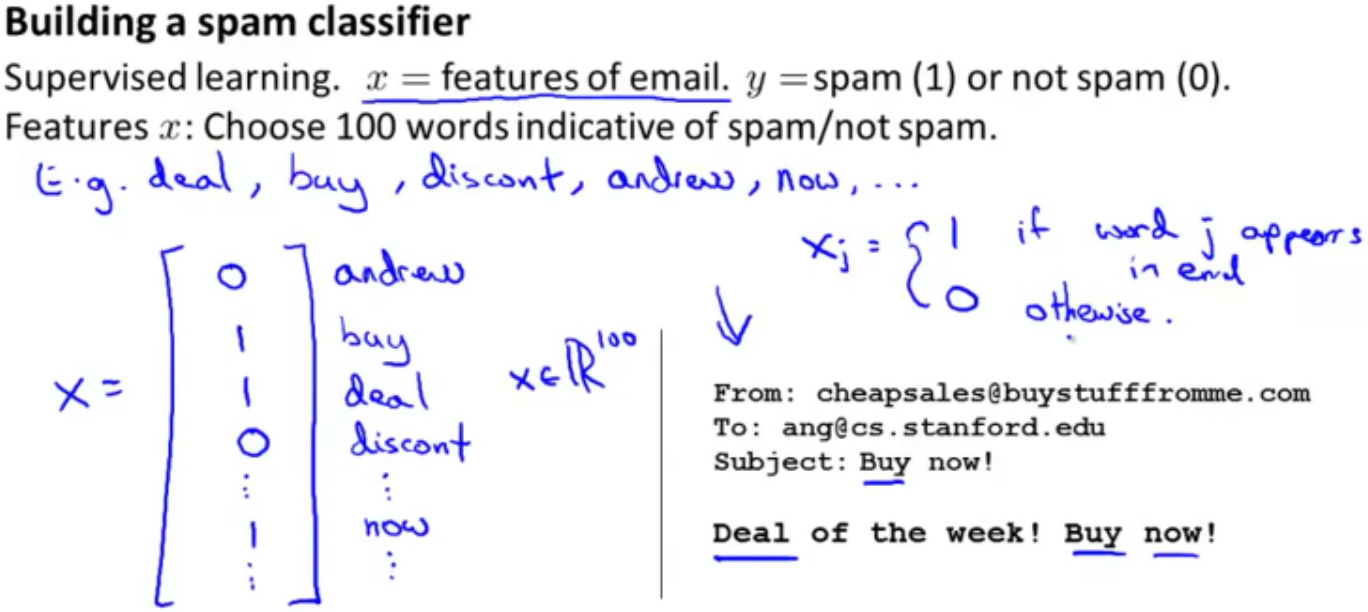
Week 6 - Lecture 2

Building a Spam Classifier

Prioritizing what to work on

System Design Example:

Given a data set of emails, we could construct a vector for each email. Each entry in this vector represents a word. The vector normally contains $10,000$ to $50,000$ entries gathered by finding the most frequently used words in our data set. If a word is to be found in the email, we would assign its respective entry a 1, else if it is not found, that entry would be a 0. Once we have all our x vectors ready, we train our algorithm and finally, we could use it to classify if an email is a spam or not.



So how could you spend your time to improve the accuracy of this classifier?

- Collect lots of data (for example "honeypot" project but doesn't always work)
- Develop sophisticated features (for example: using email header data in spam emails)
- Develop algorithms to process your input in different ways (recognizing misspellings in spam).

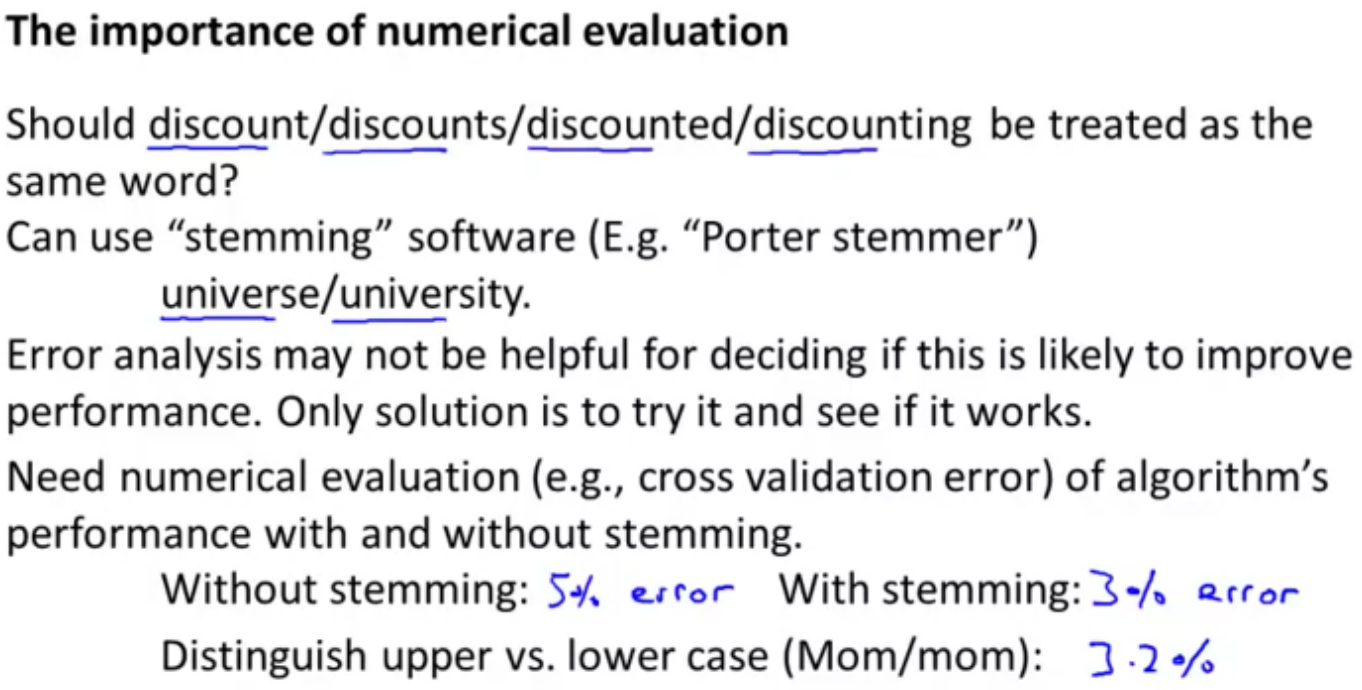
It is difficult to tell which of the options will be most helpful.

Error analysis

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.

Error Metrics for Skewed Classes

> **Example:**
> Train logistic regression model $h_\theta(x)$. ($y = 1$ if cancer, $y = 0$ otherwise.)
> Find that you got 1% error on test set. (99% correct diagnoses)
> **Only 0.50% of patients have cancer**

In that example, our 1% error rate is not so impressive anymore. To see that, consider a function like this:

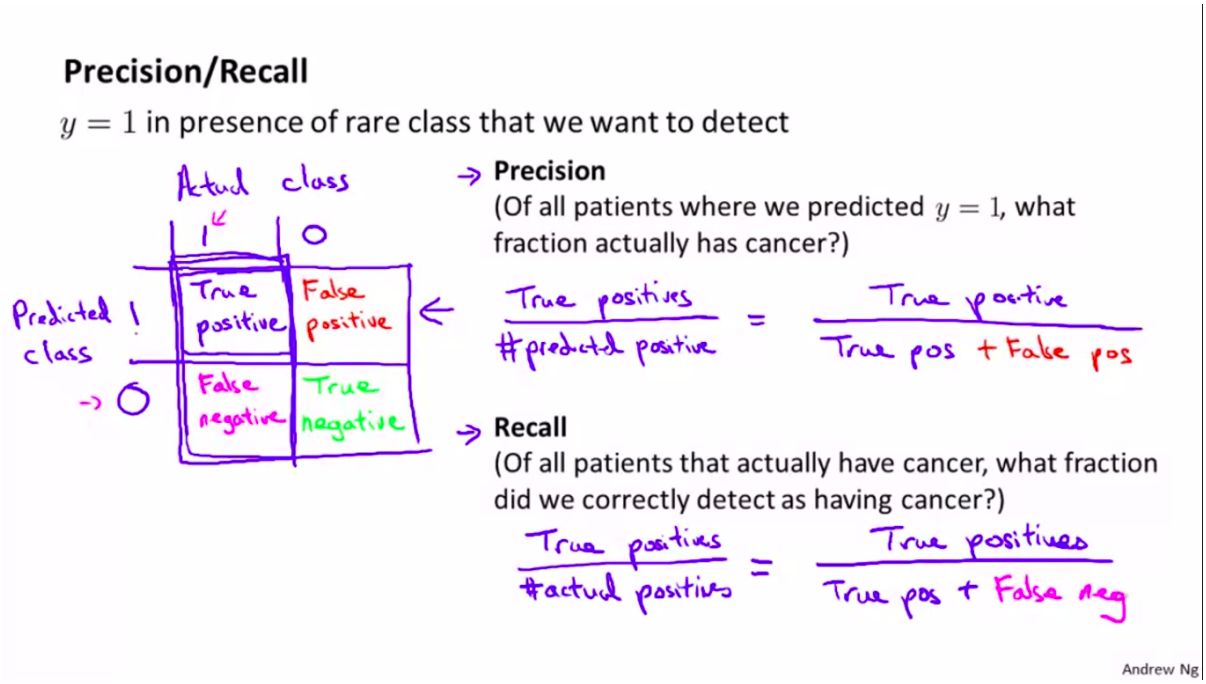
```
function y = predictCancer(x)
    y = 0; %ignore x
return
```

If we completely ignore x and just set our prediction to $y = 0$, our error rate would be 0.50% (smaller than the error of our logistic regression model).

When we have a lot more observations of one class over another, we say that we have a **skewed class** in our sample. In this case, it is harder to just use a classification accuracy metric to determine wether or not our model is improving because of relevant features or just because we added a piece of code that makes it predict more of the prominent class (like the function exemplified above).

In other words, it could be the case that improving accuracy is not assotiated with a real improvement to the model.

Precision / Recall



By attempting to increase both metrics, we can be more certain that our model is performing well because it has actually learned relevant characteristics of our problem. It would be very unlikely to obtain high precision and high recall just by using a function similar to the one exemplified in this section (guessing y=1 or y=0 for all examples in our data set).

Trading off Precision and Recall

> **Example:**
>
> Logistic regression: $0 \leq h_\theta(x) \leq 1$
> Predict 1 if $h_\theta(x) \geq 0.5$
> Predict 0 if $h_\theta(x) < 0.5$

Suppose we want to predict $y = 1$ (cancer) only if we are very confident. One way to do this is to set the threshold value to 0.7, i.e., we now are going to predict 1 if $h_\theta(x) \geq 0.7$, and 0 otherwise. In this situation, we would have:

- **Higher precision:** a higher fraction of the patients we predicted as having cancer would really have it (as we're predictiong more confidently).
- **Lower recall:** of all the patients that really have cancer, we would be predicting corectly a smaller fraction of it (because we're only predicting $y = 1$ in the cases we're more certain about it).

Aditionally, if we set our threshold value to 0.9, we would have even higher precision and lower recall.

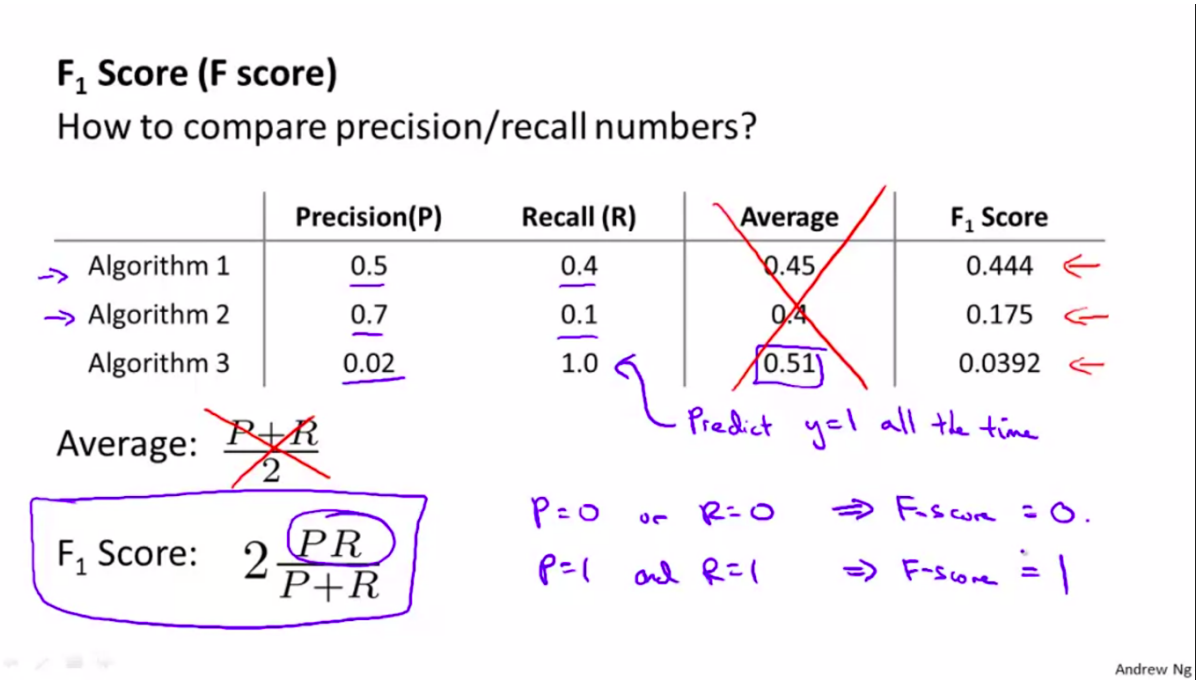
Now, suppose we want to avoid missing too many cases of cancer (avoid false negatives), i.e., when in doubt, we want to predict that he/she does have cancer. (predicting a patient doesn't have cancer when he/she actually have would be dangerous, as he/she would not seek any form of treatment). This would be equivalent to set our threshold value to, say, 0.3. In that case, we would have:

- **Smaller precision:** a higher fraction of the patients we flagged as having cancer would not actually have it.
- **Higher recall:** we would be correctly flagging a higher fraction of the patients who have cancer.

Naturally, given this situation, we come up with the following question:

Is there a way to automatically choose this threshold value? In other words, how do we decide which algorithm is best?

Answer: choose the threshold value that maximizes the F1 score over the cross-validation set.



Large Data

Designing a high accuracy learning system

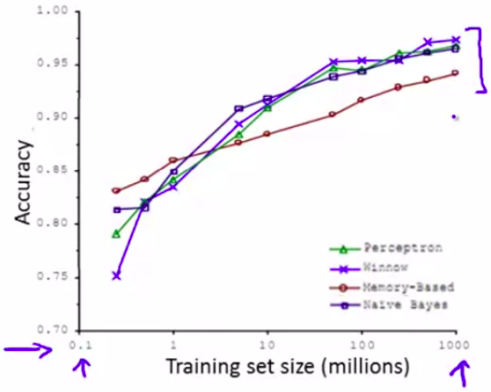
E.g. Classify between confusable words.

{to, two, too}, {then, than}

For breakfast I ate two eggs.

Algorithms

- Perceptron (Logistic regression)
- Winnow
- Memory-based
- Naïve Bayes



"It's not who has the best algorithm that wins.
It's who has the most data."

[Banko and Brill, 2001]

Large data rationale

Assume feature $x \in \mathbb{R}^{n+1}$ has sufficient information to predict y accurately.

Example: For breakfast I ate two eggs.

Counterexample: Predict housing price from only size (feet^2) and no other features.

Useful test: Given the input x, can a human expert confidently predict y?

Large data rationale

Use a learning algorithm with many parameters (e.g. logistic regression/linear regression with many features; neural network with many hidden units). low bias algorithms.

→ $J_{train}(\theta)$ will be small.

Use a very large training set (unlikely to overfit)

→ $J_{train}(\theta) \approx J_{test}(\theta)$

$J_{test}(\theta)$ will be small

References

[1] Machine Learning - Stanford University (https://www.coursera.org/learn/machine-learning)