MACHINE LEARNING SYSTEM DESIGN

Lecture 66 : Prioritizing What to Work On

Suppose we want to build a spam classifier & given an email, it flags it as spam/not spam.

spam → $y = 1$; not spam → $y = 0$

How would we choose our features x? Here's one possibility: choose 100 words indicative of spam/not spam. For instance:

1. deal → spam (likely)
2. buy → spam (likely)
3. discount → spam (likely)
4. Andrew → NOT spam (spam emails won't usually have your name in the body)
⋮
100. now → spam (likely; spam emails like to create a sense of urgency).

Given an email, we map it into a 100-dimensional feature vector by seeing if it contains these words:

$$x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

1 contains "deal"
1 contains "buy"
1 contains "discount"
0 doesn't contain "Andrew"
⋮
1 contains "now"

Since we're using supervised learning, this email is mapped to $y = 1$ if it's spam, and $y = 0$ otherwise.

In practice, what's often done is to take the most frequently occurring n words (typically 10,000 to 50,000), rather than manually picking 100 words, from the emails in the training set.

What would be the best use of time if our goal is to have a low-error classifier? Here are some options:

* Collect lots of data
  - → We already know that collecting more data doesn't always help, for instance if our learning algorithm has low bias. In many cases, though, it would help.
  - → In the spam-classification world, there are dedicated projects, called "Honey pot" projects, for collecting spam emails.
* Develop sophisticated features based on email routing information (from email header). Spammers often spoof fields in the header in order to hide their identity, or try to make the email more legitimate than it is.
* Develop sophisticated features for message body.
  - → Should "discount" and "discounts" be treated as the same word?
  - → Should "deal" and "Dealer" be treated as the same word?
* Develop sophisticated algorithm to detect misspellings.
  - → m0rtgage, med1cine, w4tches, etc. This is done by spammers so that these words aren't flagged by algorithms.

For some learning applications, it is possible to imagine coming up with many different features, but it can be hard to guess in advance which features will be most helpful. Andrew says many people just randomly fixate on one feature, based mostly on "gut feeling" and end up wasting a lot of time. There is a more systematic way of going about this, which will be the topic of the next few lectures.

Lecture 67: Error Analysis

Here's the recommended approach for tackling a complex learning application, where it's not clear in advance what may be the most relevant features:

* Start with a simple algorithm that you can implement quickly. Implement it and test it on your cross-validation data. (Andrew says he doesn't spend more than 1 day getting the first quick & dirty version working.)
* Plot learning curves to decide if more data, more features, etc. are likely to help.
    - ⇒ This is a good approach because it avoids premature optimization, i.e fixating on one thing over another, without any evidence.
* Error analysis: Manually examine the examples (in the cross-validation set) that your algorithm made errors on. See if you spot any systematic trend on what type of examples it is making errors on.

Let's go through an example of error analysis, using our spam classification as the context. Suppose we have 500 examples in the cross-validation set and our algorithm has misclassified 100 emails. We can manually examine the 100 errors and categorize them based on:

(i) What type of email is it?

- Pharma: 12 → # of emails trying to sell drugs
- Replica/fake: 4 → # of emails trying to sell fake handbags, etc.
- Steal passwords: 53
- Other: 31

It looks like we should focus on emails which try to steal passwords.

(ii) What cues (features) would have helped the algorithm classify them correctly?

Focusing on the phishing emails, we see they have the following features:

- Deliberate misspellings: 5
  (e.g. m0rtgage, med1cine, etc)
- Unusual email routing: 16
- Unusual punctuation: 32
  (e.g. too many exclamation marks)

It's clear that it would be a much better use of our time to deal with unusual punctuation than deliberate misspellings.

Note that we should perform error analysis on the cross-validation set, and not the test set. The reason is that if we develop new features by examining the test set, 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 our algorithm will generalize to new examples.

Importance of numerical evaluation

It's often useful to have a single number which gives us a good idea of how the algorithm's behaviour changes, i.e. whether it's doing better or worse and by how much, when we tweak it. For instance, in our spam-classification algorithm, should we treat discount/discounts/discounting as the same word? It would be hard to answer this by going over our training set manually, i.e by doing error analysis. The easiest way is to just try it (for instance by using "stemming" software, one example of which is "Porter Stemmer") and see if it works. If we have a numerical evaluation method of measuring our algorithm's performance (e.g. the cross-validation error), we can make an informed decision:

Without stemming: 5% With stemming: 3% ✓

The cross-validation error $J_{cv}(\theta)$ is often a natural candidate for measuring the algorithm's performance when we're experimenting with feature selection, etc. We will later see that there are cases where $J_{cv}(\theta)$ needs to be modified, for instance in the case of skewed data sets (e.g. 99% $y=0$ & 1% $y=1$).

Lecture 68: Error Metrics for Skewed Classes

Consider the problem of cancer classification, where we're using logistic regression. After training our algorithm, we find 1% error on the test set, i.e. 99% correct diagnoses. This seem pretty impressive. What if we now notice that only 0.5% of patients have cancer? Then our prediction is no longer impressive. Why? Because if my model is simply $h_\theta(x)=0$, regardless of x, I would get a 0.5% accuracy. Obviously, this is a terrible model, which just says that nobody has cancer. Perhaps the problem is using $J_{test}(\theta)$ as our error metric.

These types of data sets, where the ratio of $y=0$ / $y=1$ examples is very large or small, are called skewed classes. Let's now talk about evaluation metrics that make sense for skewed classes.

Precision / Recall

Suppose $y=1$ is the rare class (e.g. presence of cancer) we'd like to detect. (This is the usual convention: $y=1$ assigned to rare class.)

Predicted class \ Actual class	1	0
1	True Positive	False Positive
0	False Negative	True Negative

→ our algorithm predicted a positive class, which in reality is negative. Hence, a false positive.

* Precision: of all patients where we predicted $y=1$, what fraction actually have cancer?

$$\text{precision} = \frac{\text{True positives}}{\#\text{ predicted positives}} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}}$$

* Recall: of all patients that actually have cancer, what fraction did we correctly detect as having cancer?

$$\text{Recall} = \frac{\text{True positives}}{\#\text{ Actual positives}} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}}$$

We would want both precision & recall to be high. For example:

→ $h_\theta(x)=1 \ \forall x \Rightarrow$ precision $= 0.5\%$, recall $= 100\%$.

→ $h_\theta(x)=0 \ \forall x \Rightarrow$ precision $=$ null!, recall $= 0\%$.

This is equivalent to having: false pos $\ll$ True pos AND False neg $\ll$ True pos

Lecture 69: Trading Off Precision and Recall

How can we tune the trade-off between precision & recall? Let's say precision is very important to us. How can we achieve higher precision? Remember that in logistic regression $h_\theta(x)$ is the probability of $y=1$. When we go from a predicted probability $h_\theta(x)$ to a concrete prediction ($y=0$ or $y=1$), we pick a threshhold. So far we've set this to 0.5:

* $h_\theta(x) \geq 0.5 \Rightarrow$ predict 1
* $h_\theta(x) < 0.5 \Rightarrow$ predict 0.

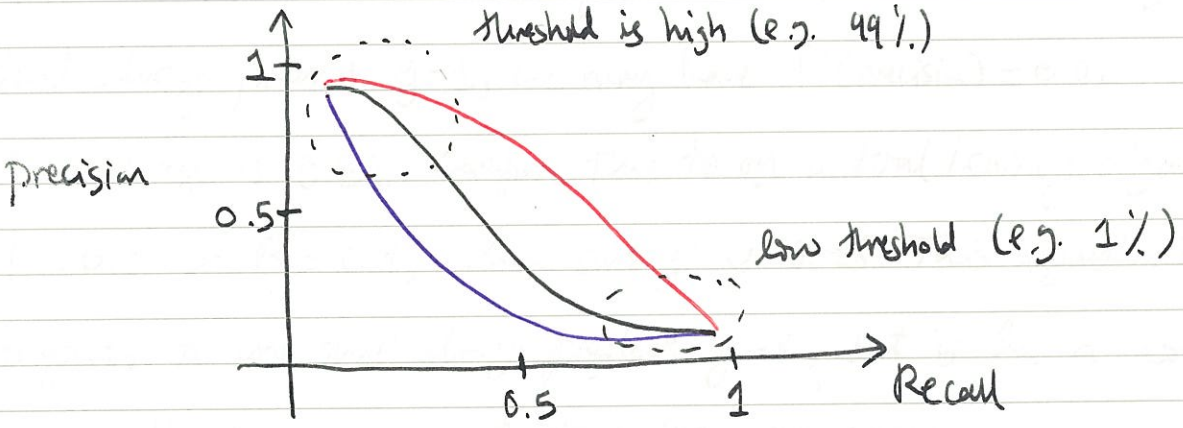
This threshhold is actually a level of confidence: we are at least 50% confidence in our predictions. What if we want to be more confident? Then we can set the threshold to a larger value. For instance, if we want to be 90% sure that the patient has cancer:

* $h_\theta(x) \geq 0.9 \Rightarrow$ predict 1
* $h_\theta(x) < 0.9 \Rightarrow$ predict 0

How would this impact precision/recall? Well, we are now making fewer predictions, most of which have high probability of being true. Therefore, precision will be higher. Recall, on the other hand, would

because we are predicting $y=1$ less frequently and we'll likely have fewer true positives as a result.

But now suppose we want to avoid missing too many cases of cancer (i.e. avoid false negatives.) In practice, false negatives are cases where the patients actually have cancer, but we're telling them they don't. Not good! In this case, we'd want to set the threshold lower, say at 30%. So we're saying that if there's a higher than 30% chance of cancer, we'll report it as cancer so that it can be looked into. This would have higher recall, but lower precision. This is the trade-off turning between precision / recall:



We generate this plot by choosing different values of the threshold, and plotting a point on the graph for each having computed precision and recall. The precise shape would be different depending on the details of the algorithm.

F_1 Score (F score)

How can we choose a reasonable value of the threshold automatically? More generally, suppose we've trained a bunch of algorithms on the training set and computed their precision & recall on the cross-validation set. How do we choose the best case? In lecture 07 we talked about the importance of having a single number that can tell us how well our algorithm is performing. In the case of skewed classes, we now have two measures: precision & recall. Can we combine them into a single number? Would the average of precision & recall work? Nope, because if we have an algorithm that always predicts $y=1$, we may have $P(\text{precision}) = 0.02$, $R(\text{recall}) = 1.0$. The average is 0.51. Compare this to an actual learning algorithm with $P = 0.5$ & $R = 0.4$, whose average is 0.45. This algorithm is clearly superior to one that always predicts $y=1$, but it has a lower score. Here's a score that makes more sense:

$$F_1 \text{ Score} = 2\frac{PR}{P+R}$$

	Precision (P)	Recall (R)	Average	F_1 Score
Algorithm 1	0.5	0.4	0.45	0.444
Algorithm 2	0.7	0.1	0.4	0.175
Algorithm 3	0.02	1.0	0.51	0.0392

There are, of course, many more possibilities for combining precision & recall. The F_1 score is one that is historically used and is popular. To get a high F_1 score, both precision & recall need to be high. To choose a threshold, we can compute the F_1 score for each case on the cross-validation set & pick the threshold with the largest score.

Lecture 70: Data for Machine Learning

How much data should we use for training? We have already cautioned against blindly collecting more data, since there are situations where an algorithm's performance will not improve with more training data (e.g. when there is high bias). However, there are situations where having a larger training set could dramatically improve the algorithm's performance, even more so that coming up with more optimized or fancier learning algorithms. Michele Banko and Eric Brill wrote a paper in 2001 which deals with this. They looked at the problem

classifying amongst confusable words: {to, two, too}, {then, than}, etc.
For example: For breakfast I ate two eggs.

They took a few supervised learning algorithms, and studied the performance of each as a function of the # of training examples. The results show that it's really the training set size that drives performance, and not the details of the algorithms. For example, at m = 10 million, the accuracies range from 87.5% → 92.5%, while for m = 1 billion, test accuracies are in the range 93.5% → 97.5%. Apparently, because of this and similar other results, there's a saying in machine learning:

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

When is this actually true?

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.

  ⇒ Counter-example: predict house price only from size and no other feature, like area.

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

Large data rationale continued

* Use a learning algorithm with many parameters (eg. logistic regression/linear regression with many features; neural network with many hidden units).

  ⇒ These are low bias algorithm. We want to avoid high bias, because in that case a large # of training examples will not help.

* Use a very large training set.

  ⇒ Even if we have a learning algorithm with many parameters, we've already seen that having a large training set acts as a regularizer & overfitting is unlikely.