1 Machine Learning System Design

1.1 Prioritizing what to work on: Spam classification example

Say you want to build a spam classifier. Spam messages often have misspelled words. We'll have a labeled training set of spam (1) and non-spam (0) messages.

Build a supervised learning classifier to distinguish between the two.

x = features of email

y = spam (1) or non-spam (0)

Features x: Choose 100 words indicative of spam/not spam.

Eg:

deal - more likely spam

buy - no spam

discount - spam

... etc

Given a piece of email, we can then encode it into a feature vector. Take the list of 100 words, sort in alphabetical order and check and see if each of those words appears in the email, 0 for doesn't occur, 1 for occurs

$$x = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 1 \\ \vdots \end{bmatrix} \begin{matrix} andrew \\ buy \\ deal \\ discount \\ \vdots \\ now \\ \vdots \end{matrix} \in \mathbb{R}^{100}$$

Each of the features $x_j = \begin{cases} 1 & if\ appears\ in\ email \\ 0 & otherwise \end{cases}$

In practice, take most frequently occuring n words (10000 to 50000) in training set, rather than manually picking 100 words.

1.1.1 Building a spam classifier

How to make it low error:

- Collect lots of data
  - e.g. "honeypot" project
- Develop sophisticated features based on email routing information (from email header)
- Develop sophisticated features for message body, e.g. should "discount" and "discounts" be treated as the same word? How about "deal" and "Dealer"? Features about punctuation?
- Develop sophisticated algorithm to detect misspellings (e.g. m0rtgage, med1cine, w4tches)

1.2 Error analysis

1.2.1 Recommender Approach

- Start with a simple algorithm that you can implement quickly. Implement it and test it on your cross-validation data.
- Plot learning curves to decide if more data, more features, etc. are likely to help
- Error analysis: Manually examine the examples (in cross validation set) that your algorithm made errors on. See if you spot any systematic trend in what type of examples it is making errors on.

Ex.:

Say you've built a spam classifier.

$m_{cv} = 500$ examples in cross validation set

Algorithm misclassifies 100 emails.

Manually examine the 100 errors, and categorize them based on:

- What type of email it is
  - pharma: 12
  - replica watches: 4
  - steal passwords/phishing: 53
  - other: 31
- What cues (features) you think would have helped the algorithm classify them correctly.
  - deliberate misspellings (m0rgage, med1cine, etc): 5
  - unusual email routing: 17
  - unusual (spamming) punctuation: 32

We want to learn which are the most difficult examples to classify. For different algorithms we often find that the same categories of examples are difficult. A quick and dirty algorithm can help identify errors and decide quickly what to work on.

1.2.2 The importance of numerical evaluation

When developing an algorithm, we should have a way to numerically evaluate an algorithm that tells how well it's doing, a single number.

Should discount/discounts/discounted/discounting be treated as the same word?

Can use "stemming" software (E.g. "Porter stemmer")

This software can help or hurt, like with universe/university.

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.

Need numerical evaluation (e.g. cross validation error) of algorithm's performance with and without stemming.

Without stemming: 5% error

With stemming: 3% error

Based on this we can decide that using stemming is a good idea.

Say we want to distinguish between upper vs lower case (Mom/mom): 3.2%

If we find that distinguising does worse than if I use only stemming, so we can decide to distinguish or not.

Manually examining those examples, is going to take a long time. Numerical method is faster.

1.3 Error metrics for skewed classes

In this context it may be tricky to come up with an error metric.

1.3.1 Cancer classification example

Train logistic regression model $h_\Theta(x)$. $\begin{cases} y = 1 & if\ cancer \\ y = 0 & otherwise \end{cases}$

We find that we have 1% error on test set.

(99% correct diagnoses)

Let's say we now find out that only 0.50% of patients have cancer. The 1% error no longer looks so impressive.

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

This algorithm would get 0.5% error. The setting of the ratio of positive to negative examples is very close to one of two extremes where in one case the number of positive examples is much, much smaller than the number of negative examples because $y = 1$ so rarely. This is what we call a problem of **skewed classes,** where we have much more examples from one class. So the problem with using classification accuracy as the evaluation metric is the following:

Say we have two learning algorithms getting:

99.2% accuracy = 0.8% error

99.5% accuracy = 0.5% error

Is the second improved over the first one or did we just replace the code with something that predicts $y = 0$ more often?

When facing skewed classes, we want to use another metric.

1.3.2 Precision/Recall

Let's say we're evaluating examples on a test set. The classes are going to be 1 or 0, binary classification. Our algorithm will predict some value for each example in the test set, 1 or 0.

		actual	class
		1	0
predicted class	1	True positive	False positive
	0	False negative	True negative

True positive: algorithm predicts 1 and it actually is 1

True negative: algorithm predicts 0 and it actually is 0

False positive: algorithm predicts 1 and it actually is 0

False negative: algorithm predicts 0 and it actually is 1

Precision: of all patients where we predicted $y = 1$ (have cancer) what fraction actually has cancer?

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

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

$$recall = \frac{True\,positives}{\#actual\,positives} = \frac{True\,positives}{True\,positives + False\,negatives}$$

If we have a learning algorithm that predicts $y = 0$ all the time, then this classifier will have $recall = 0$. A classifier with high precision and high recall is a good classifier.

We use the convention that $y = 1$ in presence of a rare class.

1.4 Trading off precision and recall

Continuing the cancer example. Let's say we trained a logistic regression classifier: $0 \leq h_\Theta(x) \leq 1$

Predict 1 if $h_\Theta(x) \geq 0.7$

Predict 0 if $h_\Theta(x) < 0.7$

Suppose we want to predict $y = 1$ (cancer) only if **very** confident. We end up with a classifier that has:

- **Higher precision**: higher fraction of patients that were diagnosed with cancer, do have cancer
- **Lower recall**: we're going to predict $y = 1$ on a lower number of patients.

Suppose we want to avoid missing too many cases of cancer (avoide false negatives).

When in doubt we want to predict $y = 1$ so the patients do get treated:

Predict 1 if $h_\Theta(x) \geq 0.3$

Predict 0 if $h_\Theta(x) < 0.3$

Well have:

- **Lower precision**: the higher fraction of the patients who we have said have cancer, the higher fraction of them will turn out not to have cancer after all.
- **Higher recall**: correctly flagging a higher fraction of patients that do have cancer.

In general there's going to be a tradeoff between precision and recall and as you vary the value of the threshold we can plot a curve [6.02]. The curve may look many different shapes.

1.4.1 $\mathbf{F_1}$ Score (F score)

Is there a way to choose this threshold automatically? If we have a few different algorithms, how do we compare different precision recall numbers?

By switching to precision/recall metric, we now have two numbers that evaluate the classifier. We still want one.

We could look at the $average = \frac{P+R}{2}$. This turns out not to be a great solution, because similar to the example we had earlier, it turns out that if we have a classifier that gives $y = 1$ all the time, then we get a very high recall. Conversely, if you have a classifier that predicts $y = 0$ almost all the time, that is it predicts $y = 1$ very sparingly, this corresponds to setting a very high threshold, and we can end up with high precision and low recall.

The $\mathbf{F_1 Score} = \mathbf{2}\frac{\mathbf{PR}}{\mathbf{P+R}}$ is a bit like taking the average of precision and recall, but gives a lower value of precision and recall. If either precision or recall is 0, the $F_1 Score = 0$ but for F_1 to be large, both precision and recall need to be large. There are more ways to combine precision and recall for F_1 score.

Measure precision and recall on cross validation set and choose the value of threshold which maximizes $2\frac{PR}{P+R}$

1.5 Data for machine learning

The issue of how much data to train on. Under certain conditions getting and training on a lot of data can can be an effective way to get good performance.

1.5.1 Designing a high accuracy learning system

Banko and Brill, 2001 experimented with effect of different algorithms on trying them out on different data set sizes. They were trying a problem of classify between confusable words:

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

For breakfast I ate _____ eggs.

Algorithms:

- Perceptron (logistic regression)
- Winnow
- Memory-based
- Naive Bayes

Most of these algorithms give similar performance and as you increase the training set size, performance almost monotonically increase. If you give an "inferior algorithm" a lot of data, it can perform well.

1.5.2 Large data rationale

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

Confusable words example: For breakfast I ate _____ eggs.

The surrouding features capture that what I want is "two" and not "to" or "too".

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

If you imagine that the house is $500 ft^2$ and no other information, there are so many other factors that affect the price of the house than just the size, it's difficult to predict the price accurately.

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

Ex.: A good English speaker can predict what words go in blank.

Suppose the features have enough information to predict the value of y.

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, can fit complex functions. Chances are that we'll be able to fit the training set well and the training error will be small.

$$J_{train}(\Theta) \textit{ will be small}$$

Let's say we use a massive training set, then even though we have a lot of parameters, then hopefully these algorithms will be unlikely to overfit.

$$J_{train}(\Theta) \approx J_{test}(\Theta)$$

So putting these two together, if training error is small and test error is close to it, then hopefully the test set error will also be small.

In order to have a well performing algorithm, the bias and variance needs to be low. Having many parameters we ensure a low bias and haring a large training set, low variance.