Developing and Evaluating an Anomaly Detection System

We're going to talk about developing a system for anomaly detection

The importance of real-number evaluation

- Previously we spoke about the importance of real-number evaluationEasier to evaluate your algorithm if it returns a single number to show if changes you made improved or worsened an algorithm's performance
    - Easier to evaluate our algorithm if it returns a single number to show if changes we made improved or worsened an algorithm's performance

When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm

To develop an anomaly detection system, would be helpful to have a way to evaluate our algorithm:

- **To evaluate our learning algorithm, we take some labeled data, categorized into anomalous and non-anomalous examples (y = 0 if normal, y = 1 if anomalous).**
    - We'll be treating anomaly detection as an unsupervised learning problem, using unlabeled data.
    - So far we've been treating anomalous detection with unlabeled data
    - If we have labeled data allows evaluation:
        - i.e. if we think something is anomalous we can be sure if it is or not.

- The process of developing and evaluating an anomaly detection algorithm is as follows:
    - Tranining set: $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ (assume normal examples/not anomalous)
        - This is our large collection of normal, non-anomalous or not anomalous examples.
        - Usually we think of this as being as non-anomalous, but it's actually okay even if a few anomalies slip into our unlabeled training set.
            - Then, take a smaller proportion of mixed anomalous and non-anomalous examples (we will usually have many more non-anomalous examples) for our cross-validation and test sets

Specific example: Aircraft engines motivating example

- 10,000 good engines (ok even if a few bad ones are here)
- 20 flawed engines (anomalous)
    - Typically when y = 1 have 20 - 50 examples (tipically range of examples)
- So, given this data set, a fairly typical way to split it into the training set, cross validation set and test set would be as follows:
    - Traning set: 60,000 good engines (y = 0)
        - We will use this to fit $p(x) = p(x_1, \mu_1, \sigma_1^2), \ldots, p(x_n, \mu_n, \sigma_n^2)$
    - CV: 2,000 good engines (y = 0), 10 anomalous (y = 1)
    - Test: 2,000 good engines (y = 0), 10 anomalous (y = 1)

- Alternative (a different way of splitting):
    - Tranining set: 6,000 good engines
    - CV: 40,000 good engines (y = 0), 10 anomalous (y = 1)
    - Test: 40,000 good engines (y = 0), 10 anomalous (y = 1)
        - Same set on CV and Test
        - **All of these are considered, less good practices and definitely less recommended.**
        - Certainly using the same data in the cross validation set and the test set, that is not considered a good machine learning practice.

To summarize we may have a set where 0.2% of the data is anomalous. We take 60% of those examples, all of which are good (y = 0) for the training set. We then take 20% of the examples for the cross-validation set (with 0.1% of the anomalous examples) and another 20% from the test set (with another 0.1% of the anomalous).

In other words, we split the data 60/20/20 training/CV/test and then split the anomalous examples 50/50 between the CV and test sets.

Algorithm Evaluation

- Fit model $p(x)$ on tranining set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$
  - We're assuming that the vast mayority examples are good/normal examples
- On a cross validation/test example x, predict

$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \varepsilon \text{ (normal)} \end{cases}$$

- We can think of algorithm a trying to predict if something is anomalous
  - But we have a label so can check
  - Makes it look like a supervised learning algorithm

Possible evaluation metrics:

- y = 0 is very common on the data set (so we have skewed classes)
  - So classification would be bad
  - An algorithm that always predicts y = 0 will have high accuracy.
- True positive, false positive, false negative, true negative
- Precision/Recall
- F_1 - Score

Can also use cross validation set to choose parameter ϵ:

- Threshold to show when something is anomalous
- A good way to choose epsilon would be to try a different values of epsilon:
  - Then pick the value of epsilon that maximizes F_1 score
  - Or that otherwise does well on our cross validation set.
- If we have CV set we can see how varying epsilon effects various evaluation metrics
- Evaluate algorithm using cross validation
- Do final algorithm evaluation on the test set

Video Question: Suppose you have fit a model $p(x)$. When evaluating on the cross validation set or test set, your algorithm predicts:

$$y = \begin{cases} 1 & \text{if } p(x) \leq \epsilon \\ 0 & \text{if } p(x) > \epsilon \end{cases}$$

Is classification accuracy a good way to measure the algorithm's performance?

- Yes, because we have labels in the cross validation / test sets.
- No, because we do not have labels in the cross validation / test sets.

> No, because of skewed classes (so an algorithm that always predicts y = 0 will have high accuracy).

- No for the cross validation set; yes for the test set.