1 Anomaly detection

1.1 Introduction

Anomaly detection is an unsupervised learning problem that has some aspects similar to supervised learning problem.

Given m normal(non-anomalous) examples $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$, we are supposed to tell whether a new example x_{test} is anomalous. The approach we will take is to build a density model $p(x)$ and choose a threshold ϵ. If $p(x_{test}) \geq \epsilon$, the new example x_{test} is flagged normal; otherwise it is recognized as an anomaly.

Anomaly detection could be used in fraud detection. Also it can be used to decide whether a product is up to the normal quality standard in manufacturing. What's more, it can be used to monitor the status of computers in a data center.

1.2 Algorithm

Suppose the training examples $x^{(i)} \in \mathbb{R}^n$. We will assume that

$$p(x) = \prod_{i=1}^{n} p(x_i; \mu_i, \sigma_i^2)$$

in which $p(x_i; \mu_i, \sigma_i^2)$ means $x_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$. We are actually taking for granted that different features x_i are independent. Although this is not always true, it does not harm the correctness of the algorithm.

First we will pick out the features x_i that might be indicative of anomalous examples. Then we fit the parameters μ_i, σ_i^2 with

$$\begin{aligned} \mu_j &= \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)} \\ \sigma_j^2 &= \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2. \end{aligned} \tag{1}$$

Now that we have the model

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{(2\pi)}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right), \tag{2}$$

we can calculate $p(x_{test})$ for any new example x_{test} and tell whether it is an anomaly.

1.3 Developing and evaluating an anomaly detection system

Assume we have some labeled data with anomalous and non-anomalous examples (y=0 normal, y=1 anomalous). In order to choose an superior anomaly

detection algorithm, we will divide the training examples into, as usual, training set, cross-validation set and test set. Note that in this case, the training set contains only normal examples, while the cv set and the test set contain both normal and anomalous examples.

In a typical case, if we have 10000 normal examples and 20 anomalies, we can put 6000 normal examples into the training set, 2000 normal exmaples and 10 anomalies into the cv set, and 2000 normal examples and 10 anomalies into the test set.

First, we will fit the model $p(x)$ with the training set $\{x^{(1)}, \ldots, x^{(m)}\}$. Then, we will make predictions for the examples in the cv set with

$$y = \begin{cases} 1 \text{ if } p(x) < \epsilon \text{ (anomaly)} \\ 0 \text{ if } p(x) \geq \epsilon \text{ (normal)} \end{cases}. \tag{3}$$

Since the data is quite skewed (most examples are normal), predication accuracy is not a good evaluation metric. We can calculate the numbers of true positive, false positive, true negative, false negative, and then calculate **Precision/Recall**, and finally evaluate the algorithm with the F_1 score.

We will choose the value of ϵ that privides the best F_1 score on the cv set. Then the algorithm can be evaluated by applying the model on the test set.

1.4 Anomaly detection v.s. supervised learning

In the section above, we labeled anomalies with $y = 1$ in order to properly evaluate the algorithm. This could be confusing since anomaly detection is an unsupervised learning algorithm, while labeling examples is what we do in supervised learning. It seems that the problem could be solved directly with supervised learning methods. Here are some guidelines helping us to decide whether to use anomaly detection or supervised learning when facing a specific problem.

1. Anomaly detection should be used when the number of positive examples is very small while the number of negative examples is large. Supervised learning should be used when there are large number of of both positive and negative examples.

2. Anomaly detection should be used when there are many "types" of anomalies, i.e. it seems unlikely that we are able to predict what future anomalies will look like based on the anomalies that have appeared. Supervised learning should be used when there are enough positive examples to help us get a sense of what an anomaly should "look like", i.e. we are confident that future anomalies will be similar to the ones we have seen.

1.5 Choosing features to use

The choice of features to be used in anomaly detection can be ambiguous and subtle.

When a feature does not "look gaussian" according to its histogram, we can make appropriate transformations to help it become gaussian. For instance, use $log(x), log(x+1), x^{1/2}, x^{1/3}$, etc instead of x.

When the current choice of features fails to differentiate an anomaly from normal examples, we should examine the anomaly and try to come up with a new feature from it and add it to the model.

A useful general principle for the choice of features in anomaly detection is to choose features that are likely to take unusually large or small values in the case of an anomaly.

1.6 Multivariate Gaussian distribution

As mentioned above, we have been taking for granted that $p(x_i)$ are independent from each other. Instead of using $p(x) = \prod_{i=1}^{n} p(x_i)$, we can model $p(x)$ all in one go with multvariate Gaussian distribution:

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(\frac{1}{2}(x-\mu)^{\mathsf{T}} \Sigma^{-1} (x-\mu)\right) \tag{4}$$

in which Σ is the covariance matrix. This is a better approach when the features chosen are correlated to each other closely.

When using multvariate Gaussian distribution, when given a training set $\{x^{(1)}, \ldots, x^{(m)}\}$, we can fit the model with

$$\begin{aligned} \mu &= \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \\ \Sigma &= \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)(x^{(i)} - \mu)^{\mathsf{T}}. \end{aligned} \tag{5}$$

When given a new example, we should calculate

$$p(x) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(\frac{1}{2}(x-\mu)^{\mathsf{T}} \Sigma^{-1} (x-\mu)\right)$$

and flag an anomaly if it turns out that $p(x) < \epsilon$.

It is not difficult to figure out that the original model used above is actually the multvariate gaussian model with

$$\Sigma = \begin{bmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_n^2 \end{bmatrix}. \tag{6}$$

When using the original model, we have to manually create extra features to capture anomalies that involve unusual combinations of existing features, while

mulvariate gaussian model automatically captures the correlations between features. However, we should be cautious to use mulvariate gaussian when there are a lot of features because it is computationally more expensive (calculation of Σ^{-1}). What's more, mulvariate gaussian requires $m > n$ to ensure the invertibility of Σ. In practice, it should only be used when m is sufficiently larger than n, e.g. $m > 10n$.

2 Recommender systems

2.1 Introduction

A recommender system aims at recommending for its users what can promisingly be of interest for them, e.g. books for customers visiting an online bookstore like Amazon, movies for a user of an online move database like IMDB, people that the user might want to add as friends on a social network like Facebook, etc. It is an important application of machine learning. It is interesting for us because it is one of the algorithms that do not call for manual choice of features.

We will illustrate the components of general recommender system with a movie recommending system as an example. The system recommends movies for users based on ratings given by all users in the database as well as the user's personal rating history. We will denote the total number of movies in the database with n_m and the total number of users with n_u. We will denote $r(i,j) = 1$ if the user j has rated movie i, and $r(i,j) = 0$ otherwise. In the case $r(i,j) = 1$, the rating of movie i given by user j will be denoted as $y^{(i,j)}$. The number of movies that have been rated by user j is denoted with $m^{(j)}$, i.e. $\sum_i r(i,j) = m^{(j)}$.

In order to make correct recommendation, the system will try to predict a user's rating for movies that he has not rated based on his personal taste, which is indicated by his rating history, as well as the category of the movie, which is indicated by other user's rating of the movie. If there are n features to describe movies, we can use an n dimension vector $\theta^{(j)}$ to represent user j's taste, and an n dimension vector $x^{(i)}$ to represent features of movie i. The rating of movie i given by user j could then be predicted with ${\theta^{(j)}}^\mathsf{T} x^{(i)}$.

Suppose we already have the $x^{(i)}$ vectors, then we can learn $\theta^{(j)}$ by minimizing the cost function

$$J(\theta^{(1)}, \ldots, \theta^{(n_u)}) = \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left({\theta^{(j)}}^\mathsf{T} x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} {\theta^{(j)}}^\mathsf{T} \theta^{(j)},$$

which could be conducted with gradient descent. Note that by convention, $\theta^{(j)}$ and $x^{(i)}$ does not contain bias component (the 0th component).

2.2 Collaborative filtering algorithm

In the previous section, we assumed that the $x^{(i)}$ vectors are already know, which is unrealistic in practice. Thus, rather than just learning $\theta^{(j)}$, we also have to learn $x^{(i)}$. Now the cost function to be minimized becomes

$$\begin{aligned} J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left({\theta^{(j)}}^{\mathsf{T}} x^{(i)} - y^{(i,j)} \right)^2 \\ + \frac{\lambda}{2} \sum_{j=1}^{n_u} {\theta^{(j)}}^{\mathsf{T}} \theta^{(j)} + \frac{\lambda}{2} \sum_{i=1}^{n_m} {x^{(i)}}^{\mathsf{T}} x^{(i)}. \end{aligned} \tag{7}$$

It can be minimized using gradient descent:

$$\begin{aligned} x^{(i)} &:= x^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} \left({\theta^{(j)}}^{\mathsf{T}} x^{(i)} - y^{(i,j)} \right) \theta^{(j)} + \lambda x^{(i)} \right) \\ \theta^{(j)} &:= \theta^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} \left({\theta^{(j)}}^{\mathsf{T}} x^{(i)} - y^{(i,j)} \right) x^{(i)} + \lambda \theta^{(j)} \right) \end{aligned} \tag{8}$$

The algorithm is called collaborative filtering algorithm. After we have learned the parameters, we can use them to predict users' ratings for movies. Note that before minimizing the cost function, we have to initialize $x^{(i)}$ and $\theta^{(j)}$ to random small values in order to break the symmetry.

The algorithm could be written in a vectorized way. We will use two matrices X and Θ defined as follows:

$$X = \begin{bmatrix} —{x^{(1)}}^{\mathsf{T}}— \\ —{x^{(2)}}^{\mathsf{T}}— \\ \dots \\ —{x^{(n_m)}}^{\mathsf{T}}— \end{bmatrix}, \Theta = \begin{bmatrix} —{\theta^{(1)}}^{\mathsf{T}}— \\ —{\theta^{(2)}}^{\mathsf{T}}— \\ \dots \\ —{\theta^{(n_u)}}^{\mathsf{T}}— \end{bmatrix}. \tag{9}$$

Then the $n_m \times n_u$ matrix that stores the prediction of the ratings of all movies by all users could simply be expressed by $X\Theta^{\mathsf{T}}$. This is called **low rank matrix factorization**. In Matlab, the vectorized implementation of the cost function and its derivatives is as follows:

```
J = 0.5 * sum(sum(((X * Theta' - Y) .* R) .^ 2)) + ...
    0.5 * lambda * (sum(sum(Theta .^ 2)) + sum(sum(X .^ 2)));
X_grad = ((X * Theta' - Y) .* R) * Theta + lambda * X;
Theta_grad = ((X * Theta' - Y) .* R)' * X + lambda * Theta;
```

With $x^{(i)}$ and $\theta^{(j)}$ learned, we can recommend unwatched movies for a specific user: simply choose the movies that he has not rated with the highest predicted ratings. We can also find movies that are the most "similar" to a given movie i: simply find movies j with the smallest $\|x^{(i)} - x^{(j)}\|$.

2.3 Mean normalization

If a user has never rated any movie, i.e. $r(i,j) = 0$ for all i, we will end up with $\theta^{(j)}$ being a vector with all elements equal to 0. As a result, the prediction for ratings of all movies by the user will be 0, which is not helpful at all for the recommendation. A reasonable choice is to use the average rating of other users as the prediction for this user. In order to accomplish this, we have to implement **mean normalization**.

For each movie, we will calculate its average rating, and store the result in vector μ. Then we will substract the average rating from all available rating results, and use the substracted results in the algorithm. Finally, the prediction of user j's rating for movie i will be ${\theta^{(j)}}^{\mathsf{T}} x^{(i)} + \mu_i$. This accomplishes mean normalization because it ensures that for a user without any rating history (thus $\theta^{(j)} = 0$), the prediction is the average rating μ_i, while the predictions for other users remain untouched.

As an example, if the rating data we have is $Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & 0 & ? & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$, then $\mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}$, and Y should be changed to $Y = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & -2.5 & ? & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$.