Beyond Supervised Learning

Week 1: # UNSUPERVISED LEARNING

Clustering

Looks at data points and finds points similar to each other. No labels! Tries to find clusters of similarity among the data.

Applications:
- market segmentation
- similar news topics
- DNA analysis
- Astronomy: galaxy grouping

K-Means Algorithm

Intuition: Randomly choose starting cluster centroids, assign points to the nearest centroid, move the centroid to avg. of its cluster. Repeat assignment & update until stable.

algorithm k-means $(x_1, \dots, x_n \in \mathbb{R}^m, k \in \mathbb{N})$:
  initialize random $\mu_1, \dots, \mu_k \in \mathbb{R}^n$
  repeat
    for $i \in [1, \dots, n]$
      $c[i] = \underset{j \in [1,k]}{\operatorname{argmin}} |x_i - \mu_j|^2$
    for $i \in [1, k]$
      $\mu_i = \text{avg}\{x_j \mid c[j] = \mu_i\}$
      or delete μ_i if $\nexists j$ s.t. $c[j] = \mu_i$

another option: → randomly re-gen. but this is common...

What if clusters are not well separated? Can still create helpful clusters...

Optimization objective: the k-means alg. is still minimizing a cost function.

Distortion Function $J(c_1, \ldots, c_m, \mu_1, \ldots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} |x_i - \mu_{c_i}|^2$

i.e. trying to minimize the avg. distance of a point to the centroid of the cluster to which it's assigned.

The algorithm repeatedly reduces J in two steps: repeat {

- shrink J by moving c_i → assign closest cluster to point
- shrink J by moving μ_{c_i} → update each cluster center

}

So each iteration should strictly reduce J and thus it will decrease J monotonically until convergence (or close enough).

Initialization. Choose $k < m = |\text{samples}|$. Init $\mu_1, \ldots, \mu_K$ to a random subset of samples.

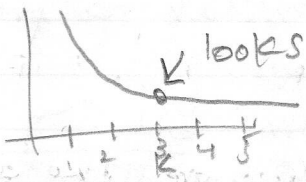
* Easy to get stuck in unfortunate local minima clusters... Can run it multiple times and pick the one w/ lowest J.

How to choose the number of clusters?

- to some extent ambiguous...
- automatically: algorithms vary. for example:

Elbow method: plot k vs. J

(note: $k = \arg\min J$ doesn't work: choose $k = m$)



- but for many applications this is a matter of judgment... choose it to maximize performance on actual target metric (i.e. application domain)

Anomaly Detection

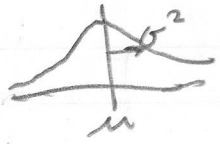
Example: identify anomalous (possibly bad) aircraft engines given manufacturing line observations and test data. Idea: determine if an engine is similar to previous ones.

Most common algorithm: Density estimation.
- build a model that predicts the $p(x)$ for new x
- choose small ε, if $p(x) < \varepsilon$, flag.

Other examples: fraud detection, cluster/DC monitoring

Gaussian (Normal) Distribution

For $x \in \mathbb{R}$, $p(x)$ is determined by μ=mean and variance=σ^2; (σ=stdev). Dist. follows a bell curve:

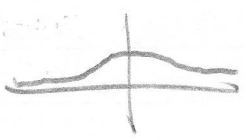


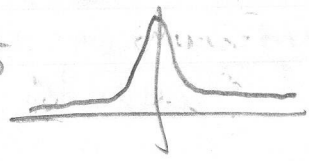
Drawing 100 numbers from a normal dist. should give a histogram roughly following the bell curve shape.

$$p(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Also recall $\int_{-\infty}^{\infty} p(x) = 1$

For fixed μ, σ determines width & height of the bell:

$\sigma^2 = 4$ 

$\sigma^2 = 0.5$ 

So given dataset $\{x_1, \ldots, x_m\}$ we need to do parameter estimation:

$$\mu = \text{avg}\{x_i\} = \frac{1}{m}\sum_i x_i$$
$$\sigma^2 = \frac{1}{m}\sum_i (x_i - \mu)^2 \quad (*)$$

(these are the MLE (maximum likelihood estimate) for μ, σ)
(sometimes $\frac{1}{m-1}$ is preferred)

For $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ given training set $\{\vec{x}_1, \ldots, \vec{x}_m\} \subseteq \mathbb{R}^n$

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

(note: this assumes the component of $\vec{x}$ are statistically independent)

where μ_i and σ_i are computed as in $(*)$ above from the training set $\{x_{1i}, \ldots, x_{mi}\}$

To summarize:
1. Choose features x_i indicative of anomaly
2. Fit params μ_i, σ_i for each feature
3. Compute $p(\vec{x})$ assuming feature independence
4. Flag as anomaly if $p(x) < \varepsilon$ small (choose this)

Evaluation

(note: we assume $y=0$ for training set)
(note: common for anom. set to be very small)

Given some labeled data of anomalous and not examples, (x, y) where x = example, $y = \{0$ if okay, 1 if anomalous$\}$.

Create train, CV, test sets.

- train: fit params assuming $y=0$
- CV: tune ε, features x_i
- test: evaluate

(note: if really very few $y=1$ examples, maybe skip test set and ¯_(ツ)_/¯)

For CV/test set eval: compute accuracy score for a classification problem. If very skewed consider confusion matrix, precision/recall, F_1.

* If you have some anomalous examples, why do anomaly detection and not just classification models? (i.e. unsupervised vs. supervised)

→ Use anomaly detection with:
- very small # of positive (anom.) examples
- many diff. types of anomalies and we may see completely new ones — a model that encodes past forms of anomaly may not recognize new ones

Anomaly detection

fraud detection
manufacturing: identify prev. unseen defects
data center monitoring

Classification

spam detection
manufacturing: identify known defects
weather prediction
disease classification

anomaly detection obviously dangerous for email filtering!

Feature selection

→ Non-gaussian features: try to transform them
- log transform $x \to \log(x + c)$
- other transforms $x \to \sqrt{x}$ etc.

Error analysis

If not performing well, look at losses.
Common: $P(x)$ common for both $y=0$ and $y=1$.
Idea: think about *what* makes the examples anomalous for the losses and encode that as a new feature.

Good to find features that take unusually large or small values in case of anomaly.
Can transform, make feature crosses