6.86x Machine Learning with Python

November 1, 2020

https://www.edx.org/course/machine-learning-with-python-from-linear-models-to

Unsupervised Learning

The objective of this section is to be able to

- understand the definition and costs in clustering
- understand the K-means algorithm and its limitations
- understand generative models and their different phases
- implement a simple EM algorithm

Clustering 1

Unsupervised learning entails methods without known patterns or results for the data, i.e. labels. Clustering is a widely-used approach for these kinds of problems. The three most important things to consider for clustering are

1. How to represent the problem as data
2. How to compare the feature vectors
3. What clustering algorithm to run for the comparison

An example of clustering could be to group similar news to someone's feed. A cleverer application could be to compress an image by reducing the number of colours it has. Each pixel would have a 24-bit RGB representation which can be used as a 24-dimensional feature vector. Then, those feature vectors can be compared and used to produce clusters of similar colours then used for compressed picture (and of course keeping a dictionary how to map the compressed pictures back to the original ones).

Clustering methods use partitions. By definition, partition of a set is a grouping of the set's elements into non-empty sets such that each element belongs to only and only one of the subsets. More precisely, $C_1, \ldots, C_K$ is a partition of $\{x_1, \ldots, x_n\}$ if and only if

$$\begin{aligned} C_1 \cup \cdots \cup C_K &= \{x_1, \ldots, x_n\} \\ C_i \cap C_j &= \emptyset, \quad \forall i \neq j \end{aligned}$$

While classification would take as an input a set of feature vectors and corresponding labels $S_n = \{x^{(i)}, y^{(i)} | i = 1, \ldots, n\}$ and a number of classes, a clustering problem requires a set of feature vectors $S_n = \{x^{(i)} | i = 1, \ldots, n\}$ only and a number K of clusters (this is subjective, but could be determined by e.g. the Elbow method `https://en.wikipedia.org/wiki/Elbow_method_(clustering)`). The output is the partitioning $C_1, \ldots C_K$ and a set of representative vectors $z_1, \ldots, z_K$ for each subset.

After the problem has been converted to suitable data representation, it is necessary to define the measure how homogeneous the feature vectors are, i.e. the cost to optimise in an ML algorithm. The cost of the whole partition can be reduced to costs of the different subsets in the partition

$$\text{cost}(C_1, \ldots, C_K) = \sum_{j=1}^{K} \text{cost}(C_j)$$

Typical cost measures include

- Subset diameter (the largest distance between two vectors in the subset)
- Average distance between all subset vectors to each other
- The sum of distance of all subset vectors to a representative vector

Furthermore, the similarity or distance can be measured with e.g. cosine similarity (insensitive to magnitude) and (squared) Euclidean distance (sensitive to magnitude).

A common algorithm to use is the K-means algorithm, which computes representative vectors $z_1, \ldots, z_K$ and assigns points to the closest z_j. Then, after the partition is done, the representative vectors are computed again to minimise some cost and a new partition is drawn. This is continued until e.g. fixed number of iterations or convergence to a local minimum. Formally, using a squared Euclidean distance,

1. Randomly select $z_1, \ldots, z_K$
2. Iterate
    2.1. Compute a partition by $\sum_{i=1}^{n} \min_{j=1,\ldots,K} \|x^{(i)} - z_j\|^2$
    2.2. Compute new representatives $z_j = \text{argmin}_z \sum_{i \in C_j} \|x^{(i)} - z\|^2$

The learning at 2.2. happens as usual; take a gradient of the new representative cost subject to z and set it to 0. This should take the next z to be the centroid of the cluster $\frac{\sum_{i \in C_j} x^{(i)}}{|C_j|}$. The real caveat is the initialisation; bad starts can lead to very suboptimal convergences to local optima. There are algorithms to ease this though, spreading the initial vectors to be further apart guided by e.g. sample variance.

See the following discussion of K-means algorithm to see pros and cons `https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages`. The other dominant clustering approach used is hierarchical clustering, where the algorithm goes through the dataset either top-down or bottom-up to produce a split of the set.

Clustering 2

Two other limitations of K-means also hinder its applicability. First, it requires a squared Euclidean distance to compute the ∇_z for the new representative vector. Furthermore, the new z is a centroid of the cluster and not a member of the set S_n. This is not ideal when we would like to e.g. present the representative vector in some use case as it is not a part of the possible options covered.

A method that should answer to these issues is called K-medoids. It is quite similar to K-means, but initialises the representative vectors from the set S_n and updates them with minimum distances in the clusters

1. Randomly select $z_1, \ldots, z_K \subseteq S_n$
2. Iterate
    2.1. Compute a partition by $\sum_{i=1}^{n} \min_{j=1,\ldots,K} \text{dist}(x^{(i)}, z_j)$

2.2. Compute new representatives $z_j = \text{argmin}_z \sum_{i \in C_j} \text{dist}(x^{(i)}, z_j), z_j \in C_j$

The K-medoids also works for data with different-sized clusters. However, it is quite clear that the computational complexity is a little higher, roughly $O(nKd)$ vs. $O(n^3Kd^2)$, where n is the sample size, K the number of clusters, and d is the dimension of the feature vectors.

Lastly, deciding the number of Ks is still a mystery. You can plot the cost with different Ks (and see the Elbow method earlier), but mostly it's subjective and based on domain knowledge. However, there might be some quantitative measures, too. Clustering is also used in preprocessing data, and e.g. in some classification tasks it can improve results significantly by running a clustering algorithms and augmenting the feature vectors with the distances of the vectors to the centroids. In that way, the best K can be chosen which minimises the classification cost.

Generative Models

Previously, we used linear separators for classification and the same decision boundaries could be made for different data if they were pretty much symmetric. This (discriminative) methodology boils down to making borders in the space based on training data topology. Another approach would be to try to understand the training data from a probabilistic perspective. Models based on this type of analysis are called generative models.
(Formally, the difference is that generative models are based on conditional probabilities of the observable $X, P(X|Y = y)$, while discriminative models are based on conditional probabilities of the target $Y, P(Y|X = x)$)

There are two main phases for building generative models: 1) trying to fit the training data to some distributions (estimation; and 2) figuring out how to use these inferences for prediction. Let's first look at two common model types, multinomial and Gaussian.

The multinomial model is quite simple. It gives the probability of any observation and assumes independence between the observations. The approach is straight-up statistical foundations where each observation has some conditional probability based on the given parameters. For example, the probability of words $w \in W$ given parameters θ

$$\begin{aligned} p(w|\theta) &= \theta_w \\ \theta_w &\geq 0, \quad \forall w \\ \sum_{w \in W} \theta_w &= 1 \end{aligned}$$

The assumption of independence is quite strong but makes things easy. For a document

D written with the words w from the vocabulary W, you could estimate the parameters θ by MLE

$$p(D|\theta) = \prod_{i=1}^{n} p(w_i|\theta) = \prod_{w \in W} \theta_w^{count(w)}$$

Take a log to reduce the equation to summation and take a gradient wrt. to the parameters θ equal to 0 to maximise the likelihood (we can use the log transformation because the function is monotonic $\rightarrow$ max of initial function must be max of the transformed one). With more than one parameter the optimisation requires Lagrange multipliers $\nabla f = \lambda \nabla g$, where $g = \sum_{w \in W} \theta_w - 1 = 0$ for the computations and $\theta_w \geq 0, \forall w$ for finding the correct solution. Then, for our above example using $L = f - \lambda g$, the derivation would be

$$\frac{\partial}{\partial \theta_w} \left[\log P(D|\theta) - \lambda (\sum_{w \in W} \theta_w - 1) \right] = 0, \quad \forall w \in W$$

which should result in the same trivial answer of $\theta_w = \frac{count(w)}{\sum_{w \in W} count(w)}$.

Prediction is quite straightforward. For each class there would be a respective set of parameters θ^k, and you'd put the thing in the class with the highest likelihood of producing the document $P(D|\theta^k)$. The P can also be multiplied with priors for each class $\pi_k = P(y = k)$, and then the posteriors should be divided with the total probability. The priors create the intercepts, and actually the MLE estimate for priors is $\pi_k = \frac{n_k}{n}$, i.e. the fraction of observations belonging to category k from the training data. See `https://stats.stackexchange.com/questions/125557/deriving-the-maximum-likelihood-for-a-generative-classification-model-for-k-clas` for more details.

For Gaussian models, there are only two parameters required: μ and σ^2. Then, the likelihood of an observation x given the parameters is

$$p(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp(-\frac{1}{2\sigma^2} \|x - \mu\|^2)$$

if the all the components are uncorrelated and have the same standard deviation μ (otherwise it's the inverse of the covariance matrix). The MLE methodology is the same, assume all observations are independent, take logs, algebra and then derivatives wrt. μ and σ set to 0. Then, the estimates should be intuitively the mean and variance of the training data.

Mixture Models; EM Algorithm

Mixture models mix together clustering and generative models and take the best of both worlds. A common mixture model would be the Gaussian mixture model (GMM), where there are 1) K clusters; 2) all normally distributed with parameters μ and σ^2; and 3) all of them having mixture weights p, i.e. a prior probability of points landing to them. This helps to refine the models to different sized clusters (relative importance) and points out the fact that dispersed points actually have some probability of belonging to almost any class. The parameters are now

$$\theta = [p_1, \ldots, p_K, \mu_1, \ldots, \mu_K, \sigma_1^2, \ldots, \sigma_K^2]$$

where we have the mixture weights for each cluster, and their corresponding d-dimensional mean vectors and variances (independent Gaussians with the same variance for each dimension per cluster $\rightarrow$ diagonal covariance matrix). Then, the likelihood of a point x would be

$$p(x|\theta) = \sum_{j=1}^{K} p_j \cdot \mathcal{N}(x; \mu_j, \sigma_j^2 I)$$

For the MLE of a training set S_n, one would have to multiply through the whole set and the points' likelihoods, and take the gradient wrt. θ. This is quite a daunting task due to interdependencies in the model and there's no closed-form solution. That's why the widely-used approach is to implement the EM-algorithm to do the job of finding a locally optimal solution. The trick is to add a probability term that the ith observation was generated by the jth cluster

$$p(j|i) = \frac{p_j \cdot \mathcal{N}(x^{(i)}; \mu_j, \sigma_j^2 I)}{p(x|\theta)}$$

Then, we can create a proxy function via the Jensen inequality (This requires some derivation, but gives a lower bound for the actual likelihood after updating θ, i.e. the real likelihood is always greater than the proxy and increases each iteration)

$$\hat{l}(S_n|\theta) = \sum_{i=1}^{n} \sum_{j=1}^{K} p(j|i) \log \left[\frac{p_j \cdot \mathcal{N}(x^{(i)}; \mu_j, \sigma_j^2 I)}{p(j|i)} \right]$$

and maximise it over the parameter set θ. The terms $p(j|i)$ have been computed beforehand and are thus fixed for the ML estimates, which are

$$\hat{p_j} = \frac{1}{n}\sum_{i=1}^{n} p(j|i)$$
$$\hat{\mu_j} = \frac{\sum_{i=1}^{n} p(j|i)x^{(i)}}{\sum_{i=1}^{n} p(j|i)}$$
$$\hat{\sigma_j^2} = \frac{\sum_{i=1}^{n} p(j|i)\|x^{(i)} - \hat{\mu}_j\|^2}{d\sum_{i=1}^{n} p(j|i)}$$

where the d in the $\hat{\sigma_j^2}$ denominator is the dimension of the input set observations x_i. The EM algorithm is then

1. Randomly select θ and compute likelihood $\hat{l}_{old}$
2. Iterate
    2.1. E-step (expectation): compute $p(j|i)$
    2.2. M-step (maximise): compute $\hat{p_j}, \hat{\mu_j}$, and $\hat{\sigma_j^2}$
    2.3. Compute $\hat{l}_{new}$ with parameters from M-step
    2.4. If $\hat{l}_{new} - \hat{l}_{old} < \epsilon$, stop. Else, $\hat{l}_{old} = \hat{l}_{new}$

Verbally, the expectation step assigns the probabilities that the observation belongs to cluster k, and the maximisation step tries to find the best parameters to fit the probability distribution generating the cluster observations. All kinds of probability distributions can be used in addition to the Gaussian presented above (of course the covariance matrix above can also be general), as long as you are able to derive the estimates for them . The initialisation plays a big role, and could be done e.g. first running K-means and then setting the parameters based on information of the representative vectors.

Summary

Unsupervised learning utilises statistical machinery to find patterns in data without pre-existing labels. Hard clustering assigns observations to different disjoint subsets called a partition. This is often done by minimising some distance measure in the clusters. When the output clusters should also be used to present something, it is better to use

a method, such as K-medoids, which provides a representative vector from the input set S_n. Other considerations include computational complexity, data density and outliers, dimensionality etc. Generative models separate from discriminatory models in a way that they try to understand how the data is generated instead of just inferring an optimal decision boundary. Generative models play effectively with conditional probabilities and utilise mostly MLE. The EM-algorithm is used to create a proxy for some difficult log-likelihood (objective) functions because they are too complex for a closed-form derivatives. The EM-algorithm provides a lower bound for the log-likelihood and converges to a local optimum, and it can be used to devise a generative model for basically any probability distribution the user can correctly write down.