UDACITY

Unsupervised Learning

Skill: 2/3

Time: 1 months

Perquisites: the previous Supervised Learning course is probably enough.

Instructor: Professor Charles Isbell (Georgia Tech), Professor Michael Littman (Brown University).

Summarized by Ido Greenberg

Contents

Randomized Optimization

1. ***Optimization***: $argmax_{x \in X} f(x)$.
2. Simple cases for optimization:
   a. Small input space X ➔ we can enumerate over all $x \in X$ and pick the maximum.
   b. Simple analytic function $f(x)$ ➔ we can use calculus to compare derivative to 0.
   c. Differentiable but complex function $f(x)$, single optimum ➔ ***Newton method***.
3. <u>Difficult optimization problem properties</u>:
   a. Big input space
   b. Complex function
   c. No/hard derivative
   d. Possibly multiple local optima
4. ***Hill climbing*** – like GD but using numerical differences rather than derivative:

```
o x = rand(X)
o while true
    ▪ y* = argmax_{y in the neighborhood of x} f(y)      # discrete space assumed
    ▪ if f(y*)>f(x)
        • x=y*     # higher point was found in the neighborhood
    ▪ else
        • break    # x is already the highest in the neighborhood
```

5. **Random restart** is a simple way to overcome local optimum, assuming there aren't too many such local optima. It reduces the sensitivity to the starting point, since multiple initial points are tried. The time cost is a constant factor.
6. The efficiency of hill climbing is strongly affected by the "attraction base" around the global optimum – the set of initial points that would converge into that optimum, rather than to other local optima.
7. ***Simulated Annealing***:
   a. <u>Inspiration</u>: a blacksmith has to heat and cool the steel iteratively until all the atoms get to the steady state of lowest energy.
   b. Doing so allows to escape local optima and explore more areas.
   c. Algorithm – Metropolis-Hastings:

```
i. x = rand(X)
ii. for i=1:N
    1. y = some arbitrary point in N(x)
    2. do x=y with prob. $P(x,y,T) = 1\ if\ f(y) \geq f(x); else\ e^{\frac{f(y)-f(x)}{T}}$
        a. If y is better we prefer it; otherwise It depends on how
           worse it is and how "hot" the temperature is.
        b. T=0 ➔ hill climbing; T=inf ➔ random walk.
    3. decrease temperature T a little
```

   d. It turns out that for every point x, $P(ending\ at\ x) = \frac{e^{f(x)/T}}{Z_T}$ (***Boltzmann distribution***):
      i. T=0 yields hill climbing, which is sensitive to local maximum; though according to the claim, it guarantees convergence to the global maximum with P=1.

        ii. I would expect that when T is high then the algorithm will do nothing beneficial; and when T becomes low the algorithm will practically act like hill climbing starting at a random point (since until this moment, is acted like a random walk).
        iii. To make this algorithm seeming less useless, I would keep the best point explored so far, and when T becomes low, I would get back to this point rather than starting from the current random point I'm at.

8. ***Genetic Algorithms*** (***GA***):
    a. <u>Inspiration</u>: as in evolution, generate multiple learners, keep the fittest ones, then generate more learners based on the previous ones, etc.
    b. "best" is determined by a ***fitness function*** defined according to the problem.
    c. Iterations in this context are called ***generations***.
    d. Possible manipulations on the parameters of the learner:
        i. ***Mutation***: change one parameter randomly.
        ii. ***Merging*** / ***Crossover***: combine parameters of one previous learner with those of another previous learner.
    e. Implementation examples:
        i. Multi-dimensional search problem: try to change only one coordinate randomly, or combine coordinates of different explored points.
        ii. NN: change certain parameters randomly, or combine matrices of different NNs.
    f. Algorithm:
        i. P_0 = initial random population of size K
        ii. While not converged
            1. Compute fitness of all $x \in P_t$
            2. Select "most fit" individuals, using either of the following:
                a. Truncate Selection: top *r* percent
                b. Roulette Wheel: weighted prob. – choose randomly with higher prob. for better individuals (as in particle filters)
            3. Generate more individuals (by mutation/merging), replacing the previous least fit individuals
    g. Crossover implementation:
        i. Choose two individuals; divide the parameters randomly into two domains; for the first child take domain A from the parent 1 and domain B from parent 2, and for the second child take domain A from the parent 2 and domain B from parent 1.
            1. Preference bias: subspaces of parameters are assumed to be beneficial by themselves; locality – parameters of the same domain "work together".
        ii. ***Uniform crossover***: similar to the first one, but instead of two domains of parameters, deal with every parameter independently – and take it randomly from parent 1 to the first child and from parent 2 to the second child, or vice versa.

            1. Preference bias: parameters are assumed to be beneficial by themselves; no locality assumed.
            2. This type of crossover actually happens biologically in the level of genes.
    h. GA tend to be great!
9. Note: the described algorithms have no significant memory – they do a lot of work, but keep only the current point/individual:
    a. No capture of history – no track on where we've already looked or what we found.
        i. TABU search keeps track of where we were and tries to avoid it again.
    b. No capture of probability distribution that maybe can be derived from the randomized search.
        i. The Annealing kind of captures probability distribution – since we know that the final point is Boltzmann-distributed.

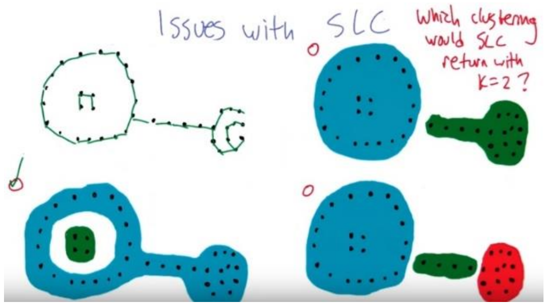
MIMIC

10. ***MIMIC***: a decades-old algorithm; name source is not explained.
11. Main idea:
    a. Model distribution directly
    b. Refine model successively
    c. Convey space structure
12. The estimated distribution in this problem is the distribution of the individuals in the space which have fitness of at least a certain level. The distribution stores the relationships between the features (e.g. coordinates) of these fit individuals.
13. High-level algorithm:
    a. set very small θ
    b. for t=1:T
        i. estimate distribution "$P_{\theta_t}(x)$"
            1. That's not trivial, and has to allow generating samples of the next desired distribution.
        ii. θ_{t+1} =nth percentile of $P_{\theta_t}(x)$
        iii. retain only $\{x|f(x) \geq \theta_{t+1}\}$
14. <u>Representation</u> (that's one possible implementation of MIMIC):
    a. ***Dependency Trees*** – Bayesian networks with up to one parent for each node (thus number of parameters is $2n_{\text{features}}-1$).
    b. $P(x) = \prod_i P\left(x_i \middle| parent(x_i)\right)$
    c. Relationships between features is captured.
    d. Sampling is easy (topological sort in a tree – linear in number of features).
15. <u>Estimating the distribution</u> through dependency trees:
    a. The estimated probability based on the graph $\hat{P}_\pi$ has to be as similar as possible to the true probability P. The difference is measured by ***KL divergence***, which can be converted to ***conditional entropy*** h and to ***mutual information*** I (all from information theory, and the last term is a symmetric measurement):

   b. $argminD_{KL}(P||\hat{P}_\pi) = argmin\sum P[lgP - lg\hat{P}_\pi] = argmin - h(P) + \sum h(x_i|\pi(x_i)) = argmin - \sum h(x_i) + \sum h(x_i|\pi(x_i)) = argmaxI(x_i;\pi(x_i))$
   c. Thus, we wish to find the dependency tree with the largest mutual information between its nodes.
   d. We begin with fully-connected graph of features (nodes) and their mutual information (edges), and we have to find the ***maximum spanning tree*** of this graph. That's a known problem in graph theory, solved by ***Prim*** and ***Kruskal*** algorithms, where the former is better for densely-connected graphs. The root of the tree can be chosen arbitrarily from the outer nodes.
   e. Every iteration of MIMIC, the mutual information is computed for the current samples, and then used to find the maximum spanning tree.
16. <u>Alternative representations</u> for MIMIC:
   a. Unconditional probability distribution (I guess it's just assuming that all features are independent, which is very easy but quite degenerated).
   b. Chain – private case of dependency tree with only one child for each node (i.e. O→O→O→O→…). In this case, the structure (order of features) is usually determined in advance (and not derived from the fully-connected graph).
   c. General Bayesian networks.
17. Dependency trees are a good compromise – they are simple Bayesian networks that do store relationships between the features but don't have exponential cost in the construction.
18. <u>Advantages</u> of MMIC:
   a. Storing structures and relationships rather than values of individual points.
   b. Representing distributions of P_θ for variant θs (not only the optimum).
   c. Local optima can do problems, but MIMIC is quite robust due to the kind of random restart applied every iteration.
   d. MIMIC typically requires much less iterations than annealing or GA, though each iteration is much more complex.
      i. Since fitness evaluations are per iteration, this is especially efficient when the computation of the fitness f(x) is expensive (e.g. the fitness of a designed rocket, which requires running a physical simulation of the rocket; or fitness requiring manual human feedback).

Clustering

1. ***Clustering*** problem:
    a. Input: data points in a metric space X (similarity may replace the metric)
        i. The distance/similarity definition expresses domain knowledge.
    b. Output: partition function
2. ***Single Linkage Clustering*** (***SLC***):
    a. Algorithm:
        i. define a cluster for each point $\{x_i\}_{i=1}^{n}$
        ii. for t=1:(n-k) % eventually k clusters remain
            1. merge the two closets clusters % distance between clusters is defined by the closest two points
    b. Tracking all the merges in a tree structure (constructed from the leaves upwards) allows going back to smaller clusters. In fact, this algorithm is very **telescopic**.
    c. Running time: **O(n^3)** for k<<n (typical), O(n^2) for k~n.
        i. Running time is derived from comparing all the pairs of points. Certain tricks may reduce the required time for that (splitting data to regions, remembering comparisons over different iterations, etc.).
    d. Preference bias: the algorithm is very **local** – it looks for the next two closest points to connect. Sequences of points may be connected in spite of being geometrically weird (see below). Random gaps in the data may leave the clusters split. This algorithm **seems well suited to *manifolds* learning**.



3. ***K-Means***:
    a. Explained in the AI course summary.
    b. Yields more compact clusters concentrated around centers.
    c. K-Means as optimization:
        i. **Configuration** – centers and partitions.
        ii. **Scores** – sum of points distances from clusters centers.
        iii. **Neighborhoods** of configurations – quite wide – all configurations with either identical centers or identical partition.
    d. K-Means is a **private case of hill climbing** – each iteration we choose the best configuration in the neighborhood.
        i. Proof of "bestness" within the neighborhood: the partition update directly minimizes the distances which define the error; and the mean

centers are the minimizers of the squared errors, as proved in least squares.

   e. Error weakly decreasing in a bounded space (finite number of points derives finite number of configurations available to the algorithm) → **convergence**.
      i. As long as the algorithm is deterministic, i.e. breaks ties in a consistent way.
      ii. The convergence is quite fast, since the neighborhoods are large.
   f. Sensitive to local optima. Solutions:
      i. Random restart.
      ii. Choosing initial centers to be data points – helps to prevent centers from being far away from the data.
      iii. Manual choice of the initial centers according to understanding of the data, kind of trying to find a big convex hole in the space of the problem.

4. ***Expectation Maximization*** (***EM***):
   a. Explained in the AI course summary.
   b. Notations:
      i. Data: $X = \{x_i\}$
      ii. Hidden variables: $Z = \{z_{ij}\} = \{P(x_i \in c_j)\}$ – updated in the E-step
      iii. Parameters: $\Theta = \{\theta_k\}$ – updated in the M-step
   c. <u>Properties</u>:
      i. Likelihood is weakly decreasing.
      ii. Never diverges, practically **converges**.
      iii. **Sensitive to local optima** – often essential to use random restart.
      iv. Any parametric distribution can be used as long as we know how to compute Expectation (Bayesian inference) and Maximization (likelihood).
      v. "Soft" partition is capable of handling overlapping (intersecting) clusters.
5. <u>Clustering properties</u>:
   a. ***Richness*** (avoid representation limits): for any assignment of objects to clusters C, there exists a distance matrix D such that the output of the clustering P_D is C.
      i. For example, constant number of clusters is not rich.
   b. ***Scale-invariance***: the output is invariant to the transformation $D \to aD$ for $a > 0$ (i.e. invariance to scales or units).
      i. For example, SLC stop-condition based on the distance between the currently closest clusters is not scale-invariant.
   c. ***Consistency***: shrinking int**ra**-cluster distances and expanding int**er**-cluster distances does not change the clustering (i.e. the notion of similarity is consistent).
      i. For example, SLC stop-condition based on the **normalized** distance between the currently closest clusters is inconsistent.
6. ***Impossibility Theorem*** (***Kleinberg***): the three desired clustering properties are mutually contradicting!

a. Thus, clustering should not be used as an automatic black-box, but rather go along with researcher’s interpretation.

Feature Selection

1. <u>Motivation</u>:
    a. **Knowledge discovery** – interpretability and insights about the data. Usually most of the data is not interesting – only few features are relevant.
    b. ***Curse of dimensionality*** – amount of needed data grows exponentially with the amount of features.
2. Given n initial features (i.e. dimensions), there are 2^n subsets of features. Finding the best feature (according to some score function f) is NP-hard.

Filtering

3. ***Filtering*** – reduce amount of features before learning.
4. Properties:
    a. Feature selection can use the data x_i and the labels y_i, but not the learner L.
    b. Separation from learning – much **faster**.
    c. Completely unsupervised – **no feedback** from learner.
5. Approaches for filtering:
    a. Keep **information gain** / variance / entropy.
    b. Keep **useful features** (detected using an independent learner).
        i. <u>Example</u>: use decision tree; take the features found by the tree and throw away the tree itself; and pass the features into a K-NN learner, which otherwise would suffer from the curse of dimensionality.
    c. Keep only independent / **non-redundant** features.

Wrapping

6. ***Wrapping*** – search features as part of the learning (search of features is "wrapped around the learning algorithm").
7. Trying to optimize the performance of the learner L while reducing as much features as possible, without going over all 2^n possible subsets.
8. Properties:
    a. Very **slow** – search and learning have to be done iteratively.
    b. Takes into account the learning model **bias**, and the learning itself – which is useful when the eventual goal is supervised, as usually.
9. Approaches:
    a. Hill climbing:
        i. Forward: start from no features and every iteration add the best feature if it's useful enough.
        ii. Backward: start from all features and every iteration remove the worst feature if it's useless enough.
    b. Randomized optimization – the previous algorithms implicitly choose the useful features, where greediness + randomization are used for efficient search.

Relevance vs. Usefulness

10. ***Relevant*** feature = informative – measured by the effect on ***BOC*** (Bayes Optimal Classifier, which is "special" since it's optimal if we look at all the possible hypotheses, see supervised course):
    a. ***Strongly relevant*** = removing it degrades BOC.
    b. ***Weakly relevant*** = not strongly relevant, but there exist a subset of features, such that removing it from the subset degrades BOC.
        i. Different features with the same information cannot be strongly relevant since each one of them can be removed. So weakly relevant can be seen as kind of non-exclusively relevant.
    c. ***Irrelevant*** = neither strongly nor weakly relevant.
11. ***Useful*** feature = helps a particular predictor to minimize error.
    a. If the general model is degenerated, then irrelevant features may be found useful (e.g. the model $w^T x > 0$ without bias b may use some constant feature $\forall i: x_i^j = 1$ to implement bias).
12. My conclusion is to keep the model rich enough, though the lecturers seem to be fine with useful-but-irrelevant features.

Feature Transformation

1. ***Feature Transformation***: pre-processing of features, generating new (typically smaller / more compact) feature set, retaining as much (relevant/useful) information as possible.
2. Note: feature selection is a private case of feature transformation.
3. Motivation:
    a. We've already seen useful implicit examples: xor, kernel methods (SVN), NN, etc.
    b. Explicit feature transformation is useful, for example, in the ad-hoc information retrieval problem ("google problem"), where you have to store lots of data with lots of dimensions, ambiguities, etc. and deal with unexpected queries.

Principal Components Analysis (PCA)

1. Well explained in Yoel's course notes.
2. In short – finds the directions (***principal components***) of maximal variance in data.
3. PCA is the linear transformation storing the **maximal amount of data per every number of coordinates**, allowing the best reconstruction: projecting onto low-dimensional space of the principal components, and then re-projecting onto the original space, yields the minimal L2 error (since that's error seems exactly like the variance of the dropped components).
4. Computed through the **eigenproblem** – which is computationally efficient (well-studied problem, and in certain algorithms the computation can be linearly shortened by taking less principal components).
5. Relevance of components can be deduced from the corresponding eigenvalue.

6. PCA may be wrong in supervised problems where the relevant information lays in certain dimensions which have quite small variance. In this sense, it is analog to **filtering**.

Independent Components Analysis (ICA)

1. While PCA maximizes variance of components, ICA maximizes statistical independence of components.
    a. Note: it was unclearly claimed that Gaussians turn out to maximize variance in data (what does it mean?), thus PCA does maximizes independence if the data consists of independent Gaussians, which is often approximately the case.
    b. Note: the sum of lots of independent variables is Gaussian, thus PCA (maximizing variance → Gaussian → sum of original variables) differs from ICA (maximizing independence → original variables), assuming the original variables themselves are far from normal.
2. Goal: transform {x}->{y}, while **vanishing the mutual information I(yi,yj)** and **maximizing data preservation I({x};{y})**.
3. Example: ***Blind Source Separation*** (***cocktail party problem***): if n people are recorded by n microphones, and the people talk independently, then ICA of the microphones recordings (dependent features) should output the original people talks (independent signals).
    a. Demonstration: http://research.ics.aalto.fi/ica/cocktail/cocktail_en.cgi
4. So it seems that **ICA is useful to reconstruct the origins of data that were mixed up, given that the origins consist of independent, non-normal signals**.
5. Note: people can separate mixed acoustic signals without multiple microphones, so ICA takes advantage of its input to efficiently solve an easier problem, yielding great results.
6. The algorithm is linear and based on mutual information. It's supposed to be explained in detail in the following frightening article: http://mlsp.cs.cmu.edu/courses/fall2012/lectures/ICA_Hyvarinen.pdf . I would rather try Wikipedia.
7. Note: both PCA and ICA try to capture the data efficiently, but they have different fundamental assumptions about how the data were generated.
8. PCA vs. ICA:

	Relations between components	**Relations with original data**	**Ordered features**	**Directionality**
PCA	Orthogonality	Best reconstruction	By variance	Non (transpose-invariant)
ICA	Independence	Max mutual information	Non	1st dim = features, 2nd dim = samples

(not sure what's the difference between best reconstruction and max mutual information, but they clearly claim these are different things)

9. They also differ in their mathematical branches:
    a. PCA – **linear algebra** – easy to understand and to implement, cheaper to run, less sensitive to local optima, well-defined and deterministic.
    b. ICA – **probability & information theory** – striving to the correct answer in probabilistic terms, but harder to find what it looks for.

10. ICA is really good at finding fundamental independent components in data, demonstrating useful **unsupervised understanding** of data:
    a. BSS – separating source signals.
    b. Faces – detecting features as noses, ears etc. (PCA finds eigenfaces).
    c. Natural scenes – finding edges in the scenes, which are the fundamental thing distinguishing between different scenes.
    d. Documents – topics.

More Algorithms

11. ***Random Components Analysis*** (***RCA***): generate random directions.
    a. Implementation: pick random projection matrix P.
    b. Surprisingly useful as filtering before classification: we reduce dimensions (avoiding curse of dimensionality) while taking all original features into account (as opposed, for example, to just removing features).
        i. "It's like splattering on my wall until it's painted, rather than systematically filling it with color".
    c. Another use may be adding features that may be useful, as in SVM.
    d. Anyway, I don't find it very useful comparing to the systematic alternatives (PCA, SVM), though it is a **faster algorithm tending to work**.
12. ***Linear Discriminant Analysis*** (***LDA***): find projection that discriminate data wrt to labels. That's actually a supervised problem, useful in spaces where effective discrimination can be linearly done. It can be seen as some linear variant of SVM, since new features are generated to allow classification.
13. ***Partial Least Squares*** (***PLS***) was not mentioned.

Information Theory

1. Fundamental terms of information theory:
    a. ***Mutual Information***: are two input vectors similar?
    b. ***Entropy***: does a certain feature carry any information?
2. Claude Shannon (Bell Labs) – "father of information theory".
3. What is information?
    a. In communication – how should we measure the amount of information passed? In words? Sentences?
    b. In physics: how much information does a thermodynamic state stores? ***Maxwell's Demon*** concept shows that energy does not necessarily physically preserved, but rather may be interchanged with information.
4. ***Entropy***: information, randomness, uncertainty, non-predictability, how many questions are required to figure out data.
5. ***Hoffman encoding***: represent common atomic signals (e.g. letters or words) with shorter sequences of bits.
    a. The encoding can be visualized as a tree.
    b. The results depend on the representation of the data – the choice of atomic signals.
6. ***Entropy***: The expected size of a message will be shorter as there is higher variance in the use of different atomic signals, since more signals will use the more frequent (and shorter) atomic signals.
    a. Thus **less randomness → less entropy → shorter messages**.
    b. $entropy = H(s) = E\left(len_{encoding}\right) = \sum P(s) \cdot len(s) = \sum P(s) \log \frac{1}{P(s)}$.
        i. It is implicitly claimed here that the most compact representation uses sequence of length 1/P(s) to represent s.
7. ***Joint entropy***: randomness contained in two variables together.
$$H(x,y) = -\sum \mathrm{P(x,y)} \log \mathrm{P(x,y)}$$
    a. If x||y, then $H(x,y) = H(x) + H(y)$.
8. ***Conditional entropy***: randomness of one variable given another.
$$H(y|x) = \sum P(x,y) log P(y|x)$$
    a. If x||y, then $H(y|x) = H(y)$.
    b. In spite of possible detection of independence, conditional entropy does not well-represent dependence (since it's not normalized wrt the unconditional entropy).
9. ***Mutual information***: reduction of randomness of one variable given another.
$$I(x,y) = H(y) - H(y|x)$$
10. ***Kullback-Leibler (KL) Divergence***: measurement of the distance between two distributions.
$$D(p||q) = \int p(x) \log \frac{p(x)}{q(x)}$$
    a. Since unsupervised learning tries to model the distribution of a dataset, measurement of distance between distributions is an important tool.
    b. It satisfies positivity (and 0 only when p=q), but not symmetry and the triangle inequality, so it's not a mathematical metric.

c. Usually p represents the true/empirical distribution, whereas q represents a hypothesis/model/approximation.