15 | Unsupervised Learning

[My father] advised me to sit every few months in my reading chair for an entire evening, close my eyes and try to think of new problems to solve. I took his advice very seriously and have been glad ever since that he did. – Luis Walter Alvarez

Learning Objectives:

- Explain the difference between linear and non-linear dimensionality reduction.
- Relate the view of PCA as maximizing variance with the view of it as minimizing reconstruction error.
- Implement latent semantic analysis for text data.
- Motivate manifold learning from the perspective of reconstruction error.
- Understand *K*-means clustering as distance minimization.
- Explain the importance of initialization in *k*-means and furthest-first heuristic.
- Implement agglomerative clustering.
- Argue whether spectral clustering is a clustering algorithm or a dimensionality reduction algorithm.

Dependencies:

If you have access to labeled training data, you know what to do. This is the "supervised" setting, in which you have a teacher telling you the right answers. Unfortunately, finding such a teacher is often difficult, expensive, or down right impossible. In those cases, you might still want to be able to analyze your data, even though you do not have labels.

Unsupervised learning is learning without a teacher. One basic thing that you might want to do with data is to **visualize** it. Sadly, it is difficult to visualize things in more than two (or three) dimensions, and most data is in hundreds of dimensions (or more). **Dimensionality reduction** is the problem of taking high dimensional data and **embedding** it in a lower dimension space. Another thing you might want to do is automatically derive a partitioning of the data into **clusters**. You've already learned a basic approach for doing this: the *k*-means algorithm (Chapter 3). Here you will analyze this algorithm to see why it works. You will also learn more advanced clustering approaches.

15.1 *K-Means Clustering, Revisited*

The *K*-means clustering algorithm is re-presented in Algorithm 15.1. There are two very basic questions about this algorithm: (1) does it converge (and if so, how quickly); (2) how sensitive it is to initialization? The answers to these questions, detailed below, are: (1) yes it converges, and it converges very quickly in practice (though slowly in theory); (2) yes it is sensitive to initialization, but there are good ways to initialize it.

Consider the question of convergence. The following theorem states that the *K*-Means algorithm converges, though it does not say how quickly it happens. The method of proving the convergence is to specify a **clustering quality** objective function, and then to show that the *K*-Means algorithm converges to a (local) optimum of that objective function. The particular objective function that *K*-Means

Algorithm 35 K-MEANS($\mathbf{D}$, K)

```
1:  for $k = 1$ to $K$ do
2:     $\boldsymbol{\mu}_k \leftarrow$ some random location          // randomly initialize mean for $k$th cluster
3:  end for
4:  repeat
5:     for $n = 1$ to $N$ do
6:        $z_n \leftarrow \text{argmin}_k \, ||\boldsymbol{\mu}_k - \boldsymbol{x}_n||$          // assign example $n$ to closest center
7:     end for
8:     for $k = 1$ to $K$ do
9:        $\boldsymbol{\mu}_k \leftarrow \text{MEAN}(\{ \boldsymbol{x}_n : z_n = k \})$          // re-estimate mean of cluster $k$
10:    end for
11: until converged
12: return $\boldsymbol{z}$          // return cluster assignments
```

is optimizing is the *sum of squared distances from any data point to its assigned center.* This is a natural generalization of the definition of a mean: the mean of a set of points is the single point that minimizes the sum of squared distances from the mean to every point in the data. Formally, the K-Means objective is:

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\mu}; \mathbf{D}) = \sum_n \left|\left| \boldsymbol{x}_n - \boldsymbol{\mu}_{z_n} \right|\right|^2 = \sum_k \sum_{n: z_n = k} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2 \tag{15.1}$$

Theorem 16 (K-Means Convergence Theorem)**.** *For any dataset* $\mathbf{D}$ *and any number of clusters K, the K-means algorithm converges in a finite number of iterations, where convergence is measured by $\mathcal{L}$ ceasing the change.*

Proof of Theorem 16. The proof works as follows. There are only two points in which the K-means algorithm changes the values of $\boldsymbol{\mu}$ or $\boldsymbol{z}$: lines 6 and 9. We will show that both of these operations can never increase the value of $\mathcal{L}$. Assuming this is true, the rest of the argument is as follows. After the first pass through the data, there are are only finitely many possible assignments to $\boldsymbol{z}$ and $\boldsymbol{\mu}$, because $\boldsymbol{z}$ is discrete and because $\boldsymbol{\mu}$ can only take on a finite number of values: means of some subset of the data. Furthermore, $\mathcal{L}$ is lower-bounded by zero. Together, this means that $\mathcal{L}$ cannot decrease more than a finite number of times. Thus, it must stop decreasing at some point, and at that point the algorithm has converged.

It remains to show that lines 6 and 9 decrease $\mathcal{L}$. For line 6, when looking at example n, suppose that the previous value of z_n is a and the new value is b. It must be the case that $||\boldsymbol{x}_n - \boldsymbol{\mu}_b|| \leq ||\boldsymbol{x}_n - \boldsymbol{\mu}_b||$. Thus, changing from a to b can only decrease $\mathcal{L}$. For line 9, consider the second form of $\mathcal{L}$. Line 9 computes $\boldsymbol{\mu}_k$ as the mean of the data points for which $z_n = k$, which is precisely the point that minimizes squared sitances. Thus, this update to $\boldsymbol{\mu}_k$ can only decrease $\mathcal{L}$. $\square$

There are several aspects of K-means that are unfortunate. First, the convergence is only to a local optimum of $\mathcal{L}$. In practice, this means that you should usually run it 10 times with different initializations and pick the one with minimal resulting $\mathcal{L}$. Second, one can show that there are input datasets and initializations on which it might take an exponential amount of time to converge. Fortunately, these cases almost never happen in practice, and in fact it has recently been shown that (roughly) if you limit the floating point precision of your machine, K-means *will* converge in polynomial time (though still only to a local optimum), using techniques of **smoothed analysis**.

The biggest practical issue in K-means is initialization. If the cluster means are initialized poorly, you often get convergence to uninteresting solutions. A useful heuristic is the **furthest-first heuristic**. This gives a way to perform a semi-random initialization that attempts to pick initial means as far from each other as possible. The heuristic is sketched below:

1. Pick a random example m and set $\boldsymbol{\mu}_1 = \boldsymbol{x}_m$.
2. For $k = 2 \dots K$:
   (a) Find the example m that is as far as possible from *all* previously selected means; namely: $m = \arg\max_m \min_{k' < k} ||\boldsymbol{x}_m - \boldsymbol{\mu}_{k'}||^2$ and set $\boldsymbol{\mu}_k = \boldsymbol{x}_m$

In this heuristic, the only bit of randomness is the selection of the first data point. After that, it is completely deterministic (except in the rare case that there are multiple equidistant points in step 2a). It is extremely important that when selecting the 3rd mean, you select that point that *maximizes* the *minimum* distance to the closest other mean. You want the point that's as far away from *all* previous means as possible.

The furthest-first heuristic is just that: a heuristic. It works very well in practice, though can be somewhat sensitive to outliers (which will often get selected as some of the initial means). However, this outlier sensitivity is usually reduced after one iteration through the K-means algorithm. Despite being just a heuristic, it is quite useful in practice.

You can turn the heuristic into an algorithm by adding a bit more randomness. This is the idea of the K-means++ algorithm, which is a simple randomized tweak on the furthest-first heuristic. The idea is that when you select the kth mean, instead of choosing the *absolute furthest* data point, you choose a data point at random, with probability proportional to its distance squared. This is made formal in Algorithm 15.1.

If you use K-means++ as an initialization for K-means, then you are able to achieve an approximation guarantee on the final value

Algorithm 36 K-MEANS++($\mathbf{D}$, K)

```
1: $\mu_1 \leftarrow x_m$ for $m$ chosen uniformly at random    // randomly initialize first point
2: for $k = 2$ to $K$ do
3:     $d_n \leftarrow \min_{k'<k} ||x_n - \mu_{k'}||^2, \forall n$    // compute distances
4:     $p \leftarrow \frac{1}{\sum_n n_d} d$    // normalize to probability distribution
5:     $m \leftarrow$ random sample from $p$    // pick an example at random
6:     $\mu_k \leftarrow x_m$
7: end for
8: run K-MEANS using $\mu$ as initial centers
```

of the objective. This doesn't tell you that you will reach the global optimum, but it *does* tell you that you will get reasonably close. In particular, if $\hat{\mathcal{L}}$ is the value obtained by running K-means++, then this will not be "too far" from $\mathcal{L}^{(\text{opt})}$, the true global minimum.

Theorem 17 (K-means++ Approximation Guarantee)**.** *The expected value of the objective returned by K-means++ is never more than $\mathcal{O}(\log K)$ from optimal and can be as close as $\mathcal{O}(1)$ from optimal. Even in the former case, with 2^K random restarts, one restart will be $\mathcal{O}(1)$ from optimal (with high probability). Formally: $\mathbb{E}\left[\hat{\mathcal{L}}\right] \leq 8(\log K + 2)\mathcal{L}^{(\text{opt})}$. Moreover, if the data is "well suited" for clustering, then $\mathbb{E}\left[\hat{\mathcal{L}}\right] \leq \mathcal{O}(1)\mathcal{L}^{(\text{opt})}$.*

The notion of "well suited" for clustering informally states that the advantage of going from $K-1$ clusters to K clusters is "large." Formally, it means that ${\mathcal{L}_K}^{(\text{opt})} \leq \epsilon^2 {\mathcal{L}_{K-1}}^{(\text{opt})}$, where ${\mathcal{L}_K}^{(\text{opt})}$ is the optimal value for clustering with K clusters, and ϵ is the desired degree of approximation. The idea is that if this condition does *not* hold, then you shouldn't bother clustering the data.

One of the biggest practical issues with K-means clustering is "choosing K." Namely, if someone just hands you a dataset and asks you to cluster it, how many clusters should you produce? This is difficult, because increasing K will always decrease ${\mathcal{L}_K}^{(\text{opt})}$ (until $K > N$), and so simply using $\mathcal{L}$ as a notion of goodness is insufficient (analogous to overfitting in a supervised setting). A number of "information criteria" have been proposed to try to address this problem. They all effectively boil down to "regularizing" K so that the model cannot grow to be too complicated. The two most popular are the Bayes Information Criteria (BIC) and the Akaike Information Criteria (AIC), defined below in the context of K-means:

$$\text{BIC:} \quad \arg\min_K \hat{\mathcal{L}}_K + K \log D \tag{15.2}$$

$$\text{AIC:} \quad \arg\min_K \hat{\mathcal{L}}_K + 2KD \tag{15.3}$$

The informal intuition behind these criteria is that increasing K is going to make $\mathcal{L}_K$ go down. However, if it doesn't go down "by enough" then it's not worth doing. In the case of BIC, "by enough" means by an amount proportional to $\log D$; in the case of AIC, it's proportional to $2D$. Thus, AIC provides a much stronger penalty for many clusters than does BIC, especially in high dimensions.

A more formal intuition for BIC is the following. You ask yourself the question "if I wanted to send this data across a network, how many bits would I need to send?" Clearly you could simply send all of the N examples, each of which would take roughly $\log D$ bits to send. This gives $N \log D$ to send all the data. Alternatively, you could first cluster the data and send the cluster centers. This will take $K \log D$ bits. Then, for each data point, you send its center as well as its deviation from that center. It turns out this will cost exactly $\hat{\mathcal{L}}_K$ bits. Therefore, the BIC is precisely measuring how many bits it will take to send your data using K clusters. The K that minimizes this number of bits is the optimal value.

15.2 *Linear Dimensionality Reduction*

Dimensionality reduction is the task of taking a dataset in high dimensions (say 10000) and reducing it to low dimensions (say 2) while retaining the "important" characteristics of the data. Since this is an unsupervised setting, the notion of important characteristics is difficult to define.

Figure 15.1: `unsup:pcadata`: Data in two dimensions

Figure 15.2: `unsup:pcadata2`: Projection of that data down to one dimension by PCA

Consider the dataset in Figure 15.1, which lives in high dimensions (two) and you want to reduce to low dimensions (one). In the case of linear dimensionality reduction, the only thing you can do is to project the data onto a vector and use the projected distances as the embeddings. Figure 15.2 shows a projection of this data onto the vector that points in the direction of *maximal variance* of the original dataset. Intuitively, this is a reasonable notion of importance, since this is the direction in which most information is encoded in the data.

For the rest of this section, assume that the data is centered: namely, the mean of all the data is at the origin. (This will simply make the math easier.) Suppose the two dimensional data is $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$ and you're looking for a vector $\boldsymbol{u}$ that points in the direction of maximal variance. You can compute this by projecting each point onto $\boldsymbol{u}$ and looking at the variance of the result. In order for the projection to make sense, you need to constrain $||\boldsymbol{u}||^2 = 1$. In this case, the projections are $\boldsymbol{x}_1 \cdot \boldsymbol{u}, \boldsymbol{x}_2 \cdot \boldsymbol{u}, \dots, \boldsymbol{x}_N \cdot \boldsymbol{u}$. Call these values $p_1, \dots, p_N$.

The goal is to compute the variance of the $\{p_n\}$s and then choose $\boldsymbol{u}$ to maximize this variance. To compute the variance, you first need to compute the mean. Because the mean of the $\boldsymbol{x}_n$s was zero, the

MATH REVIEW | EIGENVALUES AND EIGENVECTORS

todo the usual...

Figure 15.3:

mean of the ps is also zero. This can be seen as follows:

$$\sum_n p_n = \sum_n \boldsymbol{x}_n \cdot \boldsymbol{u} = \left(\sum_n \boldsymbol{x}_n \right) \cdot \boldsymbol{u} = \boldsymbol{0} \cdot \boldsymbol{u} = \boldsymbol{0} \tag{15.4}$$

The variance of the $\{p_n\}$ is then just $\sum_n p_n^2$. Finding the optimal $\boldsymbol{u}$ (from the perspective of variance maximization) reduces to the following optimization problem:

$$\max_{\boldsymbol{u}} \quad \sum_n (\boldsymbol{x}_n \cdot \boldsymbol{u})^2 \quad \text{subj. to} \quad ||\boldsymbol{u}||^2 = 1 \tag{15.5}$$

In this problem it becomes apparent why keeping $\boldsymbol{u}$ unit length is important: if not, $\boldsymbol{u}$ would simply stretch to have infinite length to maximize the objective.

It is now helpful to write the collection of datapoints $\boldsymbol{x}_n$ as a $N\times D$ matrix $\mathbf{X}$. If you take this matrix $\mathbf{X}$ and multiply it by $\boldsymbol{u}$, which has dimensions $D\times 1$, you end up with a $N\times 1$ vector whose values are exactly the values $\boldsymbol{p}$. The objective in Eq (15.5) is then just the squared norm of $\boldsymbol{p}$. This simplifies Eq (15.5) to:

$$\max_{\boldsymbol{u}} \quad ||\mathbf{X}\boldsymbol{u}||^2 \quad \text{subj. to} \quad ||\boldsymbol{u}||^2 - 1 = 0 \tag{15.6}$$

where the constraint has been rewritten to make it amenable to constructing the Lagrangian. Doing so and taking gradients yields:

$$\mathcal{L}(\boldsymbol{u}, \lambda) = ||\mathbf{X}\boldsymbol{u}||^2 - \lambda \left(||\boldsymbol{u}||^2 - 1 \right) \tag{15.7}$$

$$\nabla_{\boldsymbol{u}} \mathcal{L} = 2\mathbf{X}^\top \mathbf{X}\boldsymbol{u} - 2\lambda \boldsymbol{u} \tag{15.8}$$

$$\implies \quad \lambda \boldsymbol{u} = \left(\mathbf{X}^\top \mathbf{X} \right) \boldsymbol{u} \tag{15.9}$$

You can solve this expression ($\lambda \boldsymbol{u} = \mathbf{X}^\top \mathbf{X}\boldsymbol{u}$) by computing the first eigenvector and eigenvalue of the matrix $\mathbf{X}^\top \mathbf{X}$.

This gives you the solution to a projection into a one-dimensional space. To get a second dimension, you want to find a new vector $\boldsymbol{v}$ on which the data has maximal variance. However, to avoid redundancy, you want $\boldsymbol{v}$ to be orthogonal to $\boldsymbol{u}$; namely $\boldsymbol{u} \cdot \boldsymbol{v} = 0$. This gives:

$$\max_{\boldsymbol{v}} \quad ||\mathbf{X}\boldsymbol{v}||^2 \quad \text{subj. to} \quad ||\boldsymbol{v}||^2 = 1, \text{ and } \boldsymbol{u} \cdot \boldsymbol{v} = 0 \tag{15.10}$$

Following the same procedure as before, you can construct a La-

Algorithm 37 PCA($\mathbf{D}$, K)

```
1: $\boldsymbol{\mu} \leftarrow$ MEAN($\mathbf{X}$)                                   // compute data mean for centering
2: $\mathbf{D} \leftarrow \left(\mathbf{X} - \boldsymbol{\mu}\mathbf{1}^\top\right)^\top \left(\mathbf{X} - \boldsymbol{\mu}\mathbf{1}^\top\right)$   // compute covariance, $\mathbf{1}$ is a vector of ones
3: $\{\lambda_k, \boldsymbol{u}_k\} \leftarrow$ top $K$ eigenvalues/eigenvectors of $\mathbf{D}$
4: return $(\mathbf{X} - \boldsymbol{\mu}\mathbf{1})\,\mathbf{U}$                          // project data using $\mathbf{U}$
```

grangian and differentiate:

$$\mathcal{L}(\boldsymbol{v}, \lambda_1, \lambda_2) = ||\mathbf{X}\boldsymbol{v}||^2 - \lambda_1 \left(||\boldsymbol{v}||^2 - 1\right) - \lambda_2 \boldsymbol{u} \cdot \boldsymbol{v} \tag{15.11}$$

$$\nabla_{\boldsymbol{u}}\mathcal{L} = 2\mathbf{X}^\top\mathbf{X}\boldsymbol{v} - 2\lambda_1\boldsymbol{v} - \lambda_2\boldsymbol{u} \tag{15.12}$$

$$\Longrightarrow \quad \lambda_1\boldsymbol{v} = \left(\mathbf{X}^\top\mathbf{X}\right)\boldsymbol{v} - \frac{\lambda_2}{2}\boldsymbol{u} \tag{15.13}$$

However, you know that $\boldsymbol{u}$ is the first eigenvector of $\mathbf{X}^\top\mathbf{X}$, so the solution to this problem for λ_1 and $\boldsymbol{v}$ is given by the *second* eigenvalue/eigenvector pair of $\mathbf{X}^\top\mathbf{X}$.

Repeating this analysis inductively tells you that if you want to project onto K mutually orthogonal dimensions, you simply need to take the first K eigenvectors of the matrix $\mathbf{X}^\top\mathbf{X}$. This matrix is often called the **data covariance matrix** because $[\mathbf{X}^\top\mathbf{X}]_{i,j} = \sum_n \sum_m x_{n,i}x_{m,j}$, which is the sample covariance between features i and j.

This leads to the technique of **principle components analysis**, or **PCA**. For completeness, the is depicted in Algorithm 15.2. The important thing to note is that the eigenanalysis only gives you the projection directions. It does not give you the embedded data. To embed a data point $\boldsymbol{x}$ you need to compute its embedding as $\langle \boldsymbol{x} \cdot \boldsymbol{u}_1, \boldsymbol{x} \cdot \boldsymbol{u}_2, \dots, \boldsymbol{x} \cdot \boldsymbol{u}_K \rangle$. If you write $\mathbf{U}$ for the $D{\times}K$ matrix of $\boldsymbol{u}$s, then this is just $\mathbf{XU}$.

There is an alternative derivation of PCA that can be informative, based on *reconstruction error*. Consider the one-dimensional case again, where you are looking for a single projection direction $\boldsymbol{u}$. If you were to use this direction, your projected data would be $\mathbf{Z} = \mathbf{X}\boldsymbol{u}$. Each Z_n gives the position of the nth datapoint along $\boldsymbol{u}$. You can project this one-dimensional data back into the original space by multiplying it by $\boldsymbol{u}^\top$. This gives you *reconstructed* values $\mathbf{Z}\boldsymbol{u}^\top$. Instead of maximizing variance, you might instead want to minimize the **reconstruction error**, defined by:

$$\left|\left|\mathbf{X} - \mathbf{Z}\boldsymbol{u}^\top\right|\right|^2 = \left|\left|\mathbf{X} - \mathbf{X}\boldsymbol{u}\boldsymbol{u}^\top\right|\right|^2 \quad \text{definition of } \mathbf{Z} \tag{15.14}$$

$$= ||\mathbf{X}||^2 + \left|\left|\mathbf{X}\boldsymbol{u}\boldsymbol{u}^\top\right|\right|^2 - 2\mathbf{X}^\top\mathbf{X}\boldsymbol{u}\boldsymbol{u}^\top \quad \text{quadratic rule} \tag{15.15}$$

$$= ||\mathbf{X}||^2 + \left|\left|\mathbf{X}\boldsymbol{u}\boldsymbol{u}^\top\right|\right|^2 - 2\boldsymbol{u}^\top\mathbf{X}^\top\mathbf{X}\boldsymbol{u} \qquad \text{quadratic rule} \tag{15.16}$$

$$= ||\mathbf{X}||^2 + ||\mathbf{X}||^2 - 2\boldsymbol{u}^\top\mathbf{X}^\top\mathbf{X}\boldsymbol{u} \qquad \boldsymbol{u} \text{ is a unit vector} \tag{15.17}$$

$$= C - 2\,||\mathbf{X}\boldsymbol{u}||^2 \qquad \text{join constants, rewrite last term} \tag{15.18}$$

Minimizing this final term is equivalent to maximizing $||\mathbf{X}\boldsymbol{u}||^2$, which is exactly the form of the maximum variance derivation of PCA. Thus, you can see that maximizing variance is identical to minimizing reconstruction error.

The same question of "what should K be" arises in dimensionality reduction as in clustering. If the purpose of dimensionality reduction is to visualize, then K should be 2 or 3. However, an alternative purpose of dimensionality reduction is to avoid the curse of dimensionality. For instance, even if you have labeled data, it might be worthwhile to reduce the dimensionality before applying supervised learning, essentially as a form of regularization. In this case, the question of an optimal K comes up again. In this case, the same criteria (AIC and BIC) that can be used for clustering can be used for PCA. The only difference is the quality measure changes from a sum of squared distances to means (for clustering) to a sum of squared distances to original data points (for PCA). In particular, for BIC you get the reconstruction error plus $K \log D$; for AIC, you get the reconstruction error plus $2KD$.

15.3 Autoencoders

TODO

15.4 Further Reading

TODO