Warning: these notes have been only lightly proofread and may contain errors.

Statistics 262
Lecture Notes

Lecturers: Sahand N. Negahban & Daniel Spielman
Scribe: Addison J. Hu

Contents

Chapter 1. Preliminaries 5
1. Duality 5
2. Convexity 8
3. Projections 8
4. Clustering: k-Means 9

Chapter 2. Matrix Factorization 10
1. Recap of Matrix Factorization 10
2. Power Method as Alternating Minimization 10
3. Soft Clustering 12
4. Sparsity 14
5. Lasso 15
6. Sparse Dictionary Learning 16
7. Topic Modeling 19
8. Non-Negative Matrix Factorization 21
9. Independent Components Analysis 21
10. Matrix Completion & the Missing Entry Problem 27

Chapter 3. Optimization 29
1. Stochastic Gradient Descent 29
2. Nestorov Acceleration 29
3. Picking Step Size 30
4. Mini-Batch Stochastic Gradient Descent 30
5. Pseudo-Hessian Methods 30
6. Weight Decay 31

CHAPTER 1

Preliminaries

1. Duality

This section draws heavily from material presented in the Duality chapter of Boyd and Vanderberghe's *Convex Optimization*. It is meant to pick out important parts of the chapter to give intuition for the arguments given in class.

1.1. Introduction.

Suppose we have the problem:

$$\begin{array}{ll} \underset{x\in\mathbf{R}}{\text{minimize}} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, i \in [m] \\ & h_j(x) = 0, j \in [p] \end{array}$$

Let us denote the optimal value as p^*. The key idea in forming the Lagrangian is that we incorporate the constraints into the objective as the weighted sum of the constraint functions. Let us consider $\mathcal{L} : \mathbf{R}^n \times \mathbf{R}^m \times \mathbf{R}^p$.

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{p} \nu_j h_j(x)$$

where the vectors λ, ν are the dual variables.

1.2. Lagrange Dual Function.

We now define the Lagrange dual function $g : \mathbf{R}^m \times \mathbf{R}^p \to \mathbf{R}$:

$$\begin{aligned} g(\lambda, \nu) &= \inf_{x\in\mathcal{D}} L(x, \lambda, \nu) \\ &= \inf_{x\in\mathcal{D}} \left(f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{p} \nu_j h_j(x) \right) \end{aligned}$$

Intuitively, for any setting of λ, ν, the dual function gives the corresponding minimum over $x \in \mathcal{D}$. Therefore, the dual function yields lower bounds on p^*; that is, for all $\lambda \geq 0, \nu$:

$$g(\lambda, \nu) \leq p^*$$

Proof. Suppose $\tilde{x}$ feasible.

$$\begin{aligned}
&\Rightarrow f_i(\tilde{x}) \le 0 \ \forall \ i \in 1, \ldots, m \\
&\quad h_j(\tilde{x}) = 0 \ \forall \ j \in 1, \ldots, p \\
&\Rightarrow L(\tilde{x}, \lambda, \nu) \le f_0(\tilde{x}) \\
&\Rightarrow g(\lambda, \nu) \le L(\tilde{x}, \lambda, \nu) \le f_0(\tilde{x})
\end{aligned}$$

□

1.2.1. *Linear Approximation Interpretation.* We can reformulate our original constrained optimization problem with hard constraints in the form of "infinity" times indicator functions for when the constraints are violated.

$$\text{minimize} \qquad f_0(x) + \infty \times \sum_{i=1}^{m} \mathbf{1}\{f_i(x) > 0\} + \infty \times \sum_{j=1}^{p} \mathbf{1}\{h_i(x) \neq 0\}$$

The Lagrangian provides a softer version of these constraints, where $f_i(x) > 0$ incurs a penalty proportional to λ_i. It is easy to show that these linear functions always underestimate the "infinity times indicator" functions; therefore, it follows that the dual function yields a lower bound on the optimal value of the original problem.

1.2.2. *Lagrange Dual Function & Conjugate Functions.*

Definition. A **conjugate** f^* of a function $f : \mathbf{R}^n \to \mathbf{R}$ is defined:

$$f^*(y) = \sup_{x \in \mathbf{dom}\, f} y^T x - f(x)$$

Consider the following (highly contrived) problem:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & x = 0 \end{array}$$

We form the Lagrangian $L(x, \nu) = f(x) + \nu^T x$, and so it follows that the dual function is given by:

$$\begin{aligned}
g(\nu) &= \inf_x f(x) + \nu^T x \\
&= -\sup_x (-\nu)^T x - f(x) \\
&= -f^*(-\nu)
\end{aligned}$$

1.3. Lagrange Dual Problem. For all (λ, ν), $\lambda \ge 0$, $g(\lambda, \nu)$ gives a lower bound on p^*. It follows that we attain the best lower bound by solving:

$$\begin{array}{ll} \underset{\lambda, \nu}{\text{maximize}} & g(\lambda, \nu) \\ \text{subject to} & \lambda \ge 0 \end{array}$$

This is the **Lagrange Dual Problem** associated with the original (primal) problem.

Definition. (λ, ν) **dual feasible** implies that $\lambda \geq 0, g(\lambda, \nu) > -\infty$ implies (λ, ν) feasible for the dual problem.

We denote (λ^*, ν^*) the dual optimal multipliers.

Remark. The dual problem is a convex optimization problem, as the objective to be maximized is concave and the constraint is convex; this is true regardless of the convexity of the primal problem.

1.3.1. *Weak Duality.*

Definition. **Weak duality** implies

$$d^* \leq p^*$$

and holds even when the primal problem is not convex.

Definition. We define the **optimal duality gap** of the original problem

$$p^* - d^*$$

1.3.2. *Strong Duality.*

Definition. **Strong duality** implies

$$d^* = p^*$$

Strong duality usually holds when the primal problem is convex, i.e.,

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, i = 1, \ldots, m \\ & Ax = b \end{array}$$

with $f_0, \ldots, f_m$ convex.

Additional *constraint qualifications* are necessary beyond convexity to establish strong duality. One such constraint qualification is **Slater's condition**.

1.3.3. *Slater's Condition.* One such version of Slater's condition is as follows:

Claim 1. *There exists $x \in$ **relint** $\mathcal{D}$ such that*

$$f_x(x) < 0, i = 1, \ldots, m, \qquad Ax = b$$

Because the constraints hold with strict inequality, such a point is called *strictly feasible*. Slater's theorem states that if Slater's condition holds and the problem is convex, then strong duality holds.

Slater's condition can be relaxed such that affine constraints need not hold with strict equality. That is, if the first k constraint functions $f_1, \ldots, f_k$ are affine, then strong duality holds provided there exists an $x \in$ **relint** $\mathcal{D}$ such that:

$$f_i(x) \leq 0, i = 1, \ldots, k, \qquad f_i(x) < 0, i = k+1, \ldots, m, \qquad Ax = b$$

1.4. Geometric Interpretation.

1.5. Saddle-point Interpretation.

1.6. Optimality Conditions.

1.6.1. *Certificate of Suboptimality and Stopping Criteria.*
1.6.2. *Complementary Slackness.*
1.6.3. *KKT Conditions.*

2. Convexity

A set C is convex if

$$x, y \in C \Rightarrow \alpha x + (1-\alpha) \in C$$

for all $\alpha \in [0,1]$.

A function f is convex if for all $x, y \in \mathbf{dom}\, f$:

$$f\left(\alpha x + (1-\alpha)y\right) \leq \alpha f(x) + (1-\alpha) f(y)$$

for all $\alpha \in [0,1]$.

3. Projections

3.1. The ℓ_2 Ball. Consider the case of projection onto the ℓ_2 ball. That is, given $y \in \mathbf{R}^2$:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \|x-y\|_2^2 \\ \text{subject to} & \|x\| \leq 1 \end{array}$$

Claim 2. *This is solved by:*

$$P_{\mathcal{X}}(y) = \begin{cases} y & \text{for } \|y\|_2 \leq 1 \\ \frac{y}{\|y\|} & \text{otherwise} \end{cases}$$

Proof. We form the Lagrangian:

$$\|x-y\|_2^2 - \mu\left(1 - \|x\|_2^2\right)$$

and observe that Slater's conditions are satisfied:

$$\min_x \max_{\mu \geq 0} \|x-y\|_2^2 - \mu\left(1 - \|x\|_2^2\right) \iff \max_{\mu \geq 0} \min_x \|x-y\|_2^2 - \mu\left(1 - \|x\|_2^2\right)$$

Taking the gradient, we have

$$x^* = \frac{y}{1+\mu}$$

and substituting this back in and taking the maximum over μ, we then have:

$$\mu = \begin{cases} 0 & \text{for } \|y\| \leq 1 \\ \|y\| - 1 & \text{otherwise} \end{cases}$$

□

4. Clustering: k-Means

The k-means problem is posed as follows: given $\{x_i\}_{i=1}^n$, $x_i \in \mathbf{R}^d$, k, we want to

$$\begin{array}{ll} \underset{\pi:[n]\to[k]}{\text{minimize}} & \sum \left\|x_i - C_{\pi(i)}\right\|_2^2 \\ \text{subject to} & C_1, ..., C_k \in \mathbf{R}^d \end{array}$$

4.1. Lloyd's Algorithm. The most widely used algorithm for solving k-means is Lloyd's Algorithm.

```
Lloyd(init, k, X)
(1) $(C_i, \pi) = \texttt{init}(X, k)$
(2) while not converged:
    (a) $C_l = \frac{\sum\{x_i : \pi(i) = l\}}{\sum \mathbf{1}\{\pi(i) = l\}}$
    (b) $\pi(i) = \arg\min_{l\in[k]} \|x_i - C_l\|_2^2$
```

4.1.1. *Algorithm Initialization.* There are many choices for algorithm initialization:

(1) Choose centers randomly.
(2) Assign clusters randomly.
(3) Choose points in the datasets as centers randomly without replacement.
(4) k-means++: choose the first center randomly, and choose subsequent centers afterward randomly with probability inversely related to distance to the closest existing center.

CHAPTER 2

Matrix Factorization

1. Recap of Matrix Factorization

In Matrix Factorization, we wish to decompose a data matrix X in some meaningful way.

1.1. Principal Components Analysis as Matrix Factorization. In Principal Components Analysis, we found the most representative linear directions. Consider $X \in \mathbf{R}^{n\times d}$. We want to express it as $X = AB$, $A \in \mathbf{R}^{n\times k}, B \in \mathbf{R}^{k\times d}$. Through PCA, we have:

$$A = U_k S_k$$
$$B = V_k^T$$

with A capturing latent information, e.g., the alignment of Democrats and Republicans across different policy dimensions.

We can solve unconstrained PCA using alternating minimization.

1.2. k-Means as Matrix Factorization. We can formulate k-means in such a fashion, for $X \in \mathbf{R}^{n\times d}$:

$$\begin{aligned} \underset{A,B}{\text{minimize}} \quad & \left\|AB^T - X\right\|_F^2 \\ \text{subject to} \quad & A \in \mathbf{R}^{d\times k}, B \in \mathbf{R}^{n\times k}, B \in \{0,1\}, \sum_j B_{ij} = 1 \end{aligned}$$

To solve this, we can use alternating minimization, which is essentially equivalent to Lloyd's algorithm; we fix one matrix and minimize with repsect to the other.

2. Power Method as Alternating Minimization

Power Method. The power method is an algorithm for computing the best rank-1 approximation of a matrix in the Frobenius norm.

$$\begin{aligned} \underset{a,b}{\text{minimize}} \quad & \left\|ab^T - X\right\|_F^2 \\ \text{subject to} \quad & a \in \mathbf{R}^d, b \in \mathbf{R}^n, \|a\|_2 = 1 \end{aligned}$$

We provide this as an example of a problem that can be solved via alternating minimization.

In general, the $\|a\|_2$ constraint is not convex, as it enforces equality rather than inequality. However, projection onto this set is much simpler, and barring the zero point, projection onto this set is unique.

Remark. Projection onto the set $C = \{v : \|v\|_2 = 1\}$:

$$P_c(y) = \hat{x} = \arg\min_{x\in C} \|x - y\|_2^2$$

is solved by:

$$\hat{x} = \begin{cases} \text{anything} & \text{for } y = 0 \\ \frac{y}{\|y\|_2} & \text{for } y \neq 0 \end{cases}$$

We now present the alternating minimization algorithm for the power method.

```
AltMin(Init, X)
  (1) $a_0, b_0 = \texttt{Init}(X)$
  (2) while not finished:
      (a) $a_{k+1} = \frac{xb_k}{\|xb_k\|}$
      (b) $b_{k+1} = x^T a_{k+1}$
```

This is true because we are solving the following optimizations:

$$a_{k+1} = \arg\min_{||x||_2=1} \frac{1}{2}\left\|ab_k^T - X\right\|_F^2$$

although the constraint is no longer convex, the KKT conditions still hold. That is,

$$\min = \min_a \max_{\mu\in\mathbf{R}} \frac{1}{2}\left\|ab_k^T - X\right\|_F^2 + \mu\left(\|a\|_2^2 - 1\right)$$

still holds. The gradient is given by:

$$\nabla_a = \left(ab_k^T - X)b_k + 2\mu a\right)$$

We set this equal to zero, which implies that:

$$a(1 + 2\mu) = \frac{xb_k}{\|b_k\|_2^2}$$
$$\Rightarrow a \propto xb_k$$

to satisfy the constraint, we thus set:

$$a_{k+1} = \frac{Xb_k}{\|Xb_k\|_2}$$

This is not rigorous. We can make this rigorous by factoring b$^{[\text{why?}]}$.

Remark. This can be interpreted as a flavor of ridge regression.

We now proceed to the b_k step. We want to solve:

$$b_{k+1} = \arg\min_b \left\| b a_{k+1}^T - X \right\|_F^2$$

We observe that the gradient is given by:

$$\left(b a_{k+1}^T - X \right) a_{k+1}$$

which implies that the optimum is given at $b_{k+1} = X a_{k+1}$.

3. Soft Clustering

Recall the k-means problem previously presented. We now relax the constraint on B such that it may take on values in the $(k-1)$-simplex. Our new problem is:

$$\begin{array}{ll} \underset{A,B}{\text{minimize}} & \left\| AB^T - X \right\|_F^2 \\ \text{subject to} & A \in \mathbf{R}^{d\times k}, B \in \mathbf{R}^{n\times k}, B \geq 0, \sum_j B_{ij} = 1 \end{array}$$

Now, let us think about how the A update can be interpreted as ridge regression (in alternating minimization for the power method). This can be solved with Projected Gradient Descent (and Lagrange Duality), or with Mirror Descent (Alternating Minimziation)

3.1. Solution through Mirror Descent.

One algorithm we saw was alternating minimization. We consider the problem:

$$\begin{array}{ll} \underset{A,B}{\text{minimize}} & \| AB - X \|_F^2 \\ \text{subject to} & A \in \mathbf{R}^{n\times k}, B \in \mathbf{R}^{k\times d}, A\mathbf{I} = 1 \end{array}$$

and observe that we can solve for each column of B independently, because the Frobenius norm decouples the problem. Therefore, applying the least squares solution to each column $B_{.i}$, we have:

$$\hat{B} = \left(A^T A \right)^{-1} A^T X$$

Because the rows of A are constrained to belong to the simplex $\mathcal{A}$, we now have:

$$\begin{array}{ll} \underset{A}{\text{minimize}} & \sum_{i=1}^{n} \| A_{i.} B - X_{i.} \|_2^2 \\ \text{subject to} & A_{i.} \in \mathcal{A} \end{array}$$

which can be solved via mirror descent[1], with updates given by:

$$A_{ij}^{k+1} = \frac{A_{ij}^k \exp\left\{-t_k B_{j.}^T \left(A_{i.}^k B - X_{i.}\right)\right\}}{\sum_j A_{ij}^k \exp\left\{-t_k B_{j.}^T \left(A_{i.}^k B - X_{i.}\right)\right\}}$$
$$t_k = \frac{c}{\sqrt{k+4}}$$

Intuitively, we take inner products between row vectors, and ask, "for a particular j, how does the j^{th} row of B correlate with the error? If there is a negative correlation, we boost up A; if there is a positive correlation, we shrink A (too much error).

Observe that if we remove the exponent, we recover vanilla gradient descent. Here, we use a multiplicative update rather than an additive update. This is a good algorithm for solving least squares when the coefficients must live on the simplex.

3.2. Solution through Projected Gradient Descent. Multiplicative weights is useful in the case where the solutions are constrained to live on the simplex. For more general constrained optimization, we can use projected gradient descent. Essentially, given a convex constraint set $\mathcal{A}$, we take a normal gradient step, and then project the result of the step onto $\mathcal{A}$.

Consider the case of relaxed k-means, where $\mathcal{A}$ is the $(k+1)$-simplex:

$$\hat{B} = \arg\min_{B \in \mathcal{A}} \left\|AB^T - X\right\|_F^2$$

That is, given a matrix of vectors A, we want to find a $\hat{B}$ such that the its columns provide convex combinations of the columns of A which represent the matrix X well.

Because this is a well-behaved convex function$^{\text{[why?]}}$the initialization doesn't really matter.

```
ProjGradDescent(Init, A, X)
  (1) $B_0 = \texttt{init}(X, A)$
  (2) while not done:
      (a) $B_{k+1} = P_{\mathcal{A}}(B_{k.} - \eta(A^T(AB_{k.} - X))^T)$
```

This is exactly the least squares gradient step, just that we have a projection onto the set $\mathcal{A}$.

3.3. Relaxing the Projection. The projection step dictates:

$$P_{\mathcal{A}}(y) = \arg\min \|x - y\|_2^2$$

subject to $x \geq 0$, $\sum_{i=1}^k x_i = 1$

We can use Lagrange multipliers to rewrite the equality constraints:

$$P_{\mathcal{A}}(y) = \arg\min_{x \geq 0} \max_{\mu \in \mathbf{R}} \frac{1}{2} \|x - y\|_2^2 + \mu \left(\sum_{i=1}^n x_i - 1\right)$$

[1]Also known as Multiplicative Weights.

We may interpret this as ridge regression in the sense that we are tuning over μ, without worrying about maximizing the objective. Thus, we can reformulate our problem as follows. We want to pick a $\mu \geq 0$ and solve:

$$\hat{B} = \arg\min_{B \geq 0} \left\| AB^T - X \right\|_F^2 + \underbrace{\sum_{i=1}^{n} \left[\mu_i \sum_j B_{ij} \right]}_{\text{Regularizer}}$$

where we can interpret the μ_i as a parameter of regularization. We recall that Ridge Regression may be formulated as:

$$\arg\min_x \frac{1}{2} \|Ax - b\|_2^2 + \frac{\lambda}{2} \|x\|_2^2$$

Basically, we want to constrain the ℓ_2 norm, but we don't want to have to go through the hassle of finding the Lagrange multiplier in closed form. We use μ as a parameter to control the sum instead. We penalize the problem for making the sum too big; previously, in soft clustering, we wanted it to be close to one.

We may now observe that

$$\hat{B} = \arg\min_{B \geq 0} \left\| AB^T - X \right\|_F^2 + \underbrace{\sum_{i=1}^{n} \left[\mu_i \sum_j B_{ij} \right]}_{\text{Regularizer}}$$

is just like Projected Gradient Descent, but we don't need to optimize over μ anymore. (Why are penalties being assigned on an observation-by-observation basis rather than a variable-by-variable basis?)[why?]

So, do we always want B to be positive? What if we want to add some signal in some directions, and remove some signal in other directions?

4. Sparsity

4.1. Relaxing Sparsity. Recall the projection step[why?]

Recalling that here $B \in \mathbf{R}^{n \times k}$, we want to solve $\hat{B}_{i.}$ to be

$$\arg\min_b \left\| Ab^T - X.i \right\|_F^2 + \mu_i \|b\|_1$$

subject to $b \geq 0$, $b \in \mathbf{R}^k$.

Now, we can use gradient descent:

$$b_{t+1} = \left\lfloor b_t - \eta_t [A^T (Ab_t - X_{.i}) + \mu \mathbf{1}_k] \right\rfloor_+$$

So, we will do a normal gradient descent step, as if we didn't care that $B \geq 0$, and if it's negative, just threshold it to zero. This performs sparsification. One of the more popular reasons for sparsity is to add intepretability. For example, in k-means, we imposed a restriction on the rows of B to be one-sparse; each point belongs to one cluster. Here, we relax the one-sparse assumption, but still enforce sparsity.

4.2. Encouraging Sparsity. In many applications, we don't want the $B \geq 0$ constraint, which gave us sparsity. If we drop the $B \geq 0$ constraint, then we get Independent Component Analysis. We also get Sparse-PCA. We still want to encourage sparsity in these settings. As opposed to PCA, ICA just wants to factorize data in some meaningful way. We want a factorization such that the columns are independent.

Example 4.1. An example is the cocktail party problem. There are n recordings of n speakers speaking simultaneously, about unrelated things. We want to decompose the sum of the voices. We want to find independent columns. One of the best ways to do this is to make B sparse.

Example 4.2. Sparse-PCA could find usage in gene expressions, where there are a lot of genes, but maybe only a hundred patients. We want to find a sparse subset of the 50,000 genes. We don't have enough patients to do PCA. So, we can encourage sparsity on A, but this is dual, so it doesn't matter (we can just consider the transpose).

5. Lasso

We now want to solve:

$$\frac{1}{2}\left\|AB^T - X\right\|_F^2$$

such that $B_{i.}$ is s-sparse. In k-means, we did one-sparse. It is much harder to do s-sparse.

One way to do this is with the lasso. We use L_1 regularization:

$$\begin{array}{ll} \underset{A,B}{\text{minimize}} & \frac{1}{2}\left\|AB^T - X\right\|_F^2 + \sum_{i=1}^{n} \mu_i \left\|B_{i.}\right\|_1 \\ \text{subject to} & A \in \mathbf{R}^{d\times k}, B \in \mathbf{R}^{n\times k} \end{array}$$

For μ_i we have many choices, if X is normalized then we can set all μ to be equal. If X is not normalized and one column is much larger than the rest, then it will dominate the optimization.

Because there are no constraints on A, that optimization is easy. We look more closely at B.

5.1. Solving for $B_{i.}$. We can solve:

$$B_{i.} = \arg\min_b \|Ab - X_{.i}\|_2^2 + \mu \|b\|_1$$

with gradient descent:

$$b_{t+1} = b_t - \eta[A^T(Ab_l - X_{.i}) + \mu \operatorname{sign}(b)]$$

where sign(0) can be anything in $(-1, 1)$ (subgradient). This converges well with $\eta_t = \frac{c}{\sqrt{t+2}}$.

5.2. Alternative Optimization Methods. There are (at least) two methods that converge much more quickly:

- Composite Objective Gradient Descent
- Coordinate Descent

We can thus instead use Iterative Soft Thresholding:

$$b_{t+1} = S_{\mu \times \eta_t}\left(b_t - \eta_t[A^T(Ab_t - X_{.i})]\right)$$

Definition. The soft-thresholding function is defined as follows:

$$S_\alpha(x) = \text{ sign}(x) \cdot \max\{|x| - \alpha, 0\}$$

That is, if we are within $(-\alpha, \alpha)$, we set it to zero; otherwise, we shrink towards zero.

6. Sparse Dictionary Learning

Recall that we want to:

$$\begin{aligned} &\underset{A,B}{\text{minimize}} && \|AB - X\|_F^2 \\ &\text{subject to} && B \in \mathcal{B} \subseteq \mathbf{R}^{k \times n}, X \in \mathbf{R}^{d \times n}, A \in \mathcal{A} \end{aligned}$$

The constraints on A, B depends on the problem (e.g., k-means, etc.). Note that here the dimensions of B are flipped!

We examine an example from neuroscience. X may be some image. A may be some way of representing the image. B denotes the neurons firing (specifically, the columns of B denotes the activity of the neurons).

Our intuition is that the columns of B are sparse (recall that each column of B denotes which columns of A we are interested in.). Each column of A denotes the quantity that the neuron that may be interested in seeing (each neuron is assigned to a column.) For example, if the columns of B are one-sparse, then each column of B denotes which column of A we should put in its place.

$$\mathcal{B} = \{B : \ \forall \ j, B_{.j} \text{ is "sparse"}\}$$

For example:

$$\mathcal{B}_0 = \left\{B : \ \forall \ j, \|B_{.j}\|_0 \leq S_j\right\}$$

where the ℓ_0 norm counts the number of nonzero entries. That is, we want to enforce S_j-sparsity in each column.

6.1. Optimization. Recall that we are solving these problems with Alternating Minimization:

$$A_{l+1} = \arg\min_{A \in \mathcal{A}} \|AB_l - X\|_F^2 \qquad (P_A)$$

$$B_{l+1} = \arg\min_{B \in \mathcal{B}} \|A_{l+1}B - X\|_F^2 \qquad (P_B)$$

So, how can we solve these subproblems, given the the constraints of $\mathcal{A}, \mathcal{B}$? If $\mathcal{A} = \mathbf{R}^{d \times k}$, then P_A is just linear regression. If $\mathcal{B} = \mathcal{B}_0$, then (in general, P_B is "hard", taking $O(nk^s)$

(because we need to cycle over all possible subsets, where $s \triangleq \max_j s_j$ In k-means, we set $s = 1$, so it is tractable. When s becomes interesting, it becomes intractable.

6.2. Easier Way to Encode Sparsity.

$$\mathcal{B}_{l_1} = \left\{B : \|B_{.j}\|_1 \leq s\right\}$$

Here we bound the sum of absolute values. The set $\mathcal{B}_{l_1}$ is now convex, so we can run projected gradient descent with convergence guarantees. Now, we have an outer loop of alternating minimization, but with P_B, we have an inner loop. This time, the projection is onto the ℓ_1 ball.

We can also consider the $C_\triangle$ (simplex):

$$\mathcal{B}_\triangle = \left\{B : B \geq 0, \sum_{i=1}^{k} B_{ij} = 1\right\}$$

We can solve this P_B problem using multiplicative weights or projected gradient descent.

6.3. Inducing Sparsity through Regularization. We can also encourage sparsity not through constraints, but instead with regularization.

$$P'_B : \arg \min_{B \in \mathbf{R}^{k\times n}} \|AB - X\|_F^2 + \sum_j \lambda_j \|B_{.j}\|_1$$

where here we denote the elementwise-equivalent of the Frobenious norm.

This is softer than with constraints. Now, we have a tradeoff between how well we can fit, against the penalty. A shrinkage in the loss function must overcome the imposed penalty. This is more popular than the $\mathcal{B}_{\ell_1}$ version of the problem.

There is a duality between the constrained problem and the penalized problem. There is a result which says that for every constraint we could impose, there is a corresponding λ^*. But a very small difference in $\mathcal{B}$ could result in a large difference in λ^*. Using the penalized problem yields more robust results.

6.4. Proximal Gradient Descent. Suppose $\lambda_j = \lambda \ \forall \ \lambda_j$.

$$B_{h+1} = S_{\eta_h \lambda}\left(B_h - \eta_h(A^T(AB_h - X))\right)$$

The outer B_{l+1} loop updates the B_l, whereas here we are iterating through B_h to solve problem P'_B. Recall the soft-thresholding function from last class:

$$S_\alpha(x) = \operatorname{sign}(x) \cdot \max\{|x| - \alpha, 0\}$$

6.5. Coordinate Descent. To solve P'_B, pick some entry of B_{ij} and optimize. For a single entry, this is very easy.

(1) Let $b = B_{.j}$.
(2) Let $x = X_{.j}$.
(3) Solve $\|Ab - x\|_2^2 + \lambda \|b\|_1$ for each j by optimizing over b_i.

We can see that this work because we can split the Frobenious norm splits across the columns, and we sum over the ℓ_1 norms of the columns.

$$\min_{b_i} \left\| Ab_{\setminus i} + A_{.i}b_i - x \right\|_2^2 + \lambda |b_i|$$

where

$$b_{\setminus i,j} = \begin{cases} b_j & \text{for } j \neq i \\ 0 & \text{otherwise} \end{cases}$$

Now, let $y = x - Ab$.

Claim 3.

$$b_i = S_{\frac{\lambda}{2\|A\|_2^2}} \left(\frac{A_{.i}^T y}{\|A_{.i}\|_2^2} \right)$$

Observe that this is exactly the least-squares solution for a single coordinate.

This is a nice, simple, closed form solution for each iterate.Incidentally, `scikit-learn` formerly used gradient descent for non-negative matrix factorization, but deprecated it in favor of coordinate descent.

Proof. Let $a = A_{.i} \in \mathbf{R}^d, y \in \mathbf{R}^d$.

We can see this from:

$$\min_{b \in \mathbf{R}} \|ab - y\|_2^2 + \lambda |b|$$

We can take the subgradient of this and set it to zero:

$$a^T(ab^* - y) + \frac{\lambda}{2} \operatorname{sign}(b^*) = 0$$

Observe that $a^T a = \|a\|_2^2$. Then we have

$$b = \frac{a^T y}{\|a\|_2^2} - \frac{\lambda}{2\|a\|_2^2} \operatorname{sign}(b)$$

So, we have on a case-by-case basis:

$$b = \begin{cases} \frac{a^T y - \frac{1}{2}\lambda}{\|a\|_2^2} & \text{for } a^T y > \frac{\lambda}{2} \\ \frac{a^T y + \frac{1}{2}\lambda}{\|a\|_2^2} & \text{for } a^T y < \frac{\lambda}{2} \end{cases}$$

Suppose $|a^T y| \leq \frac{\lambda}{2}$. Then $b = 0,\ \operatorname{sign}(0) = \frac{2a^T y}{\lambda}$. We can check that $\left| \frac{2a^T y}{\lambda} \right| \leq 1$ as required for $\frac{d|x|}{dx}$. □

6.6. Block Coordinate Descent. There is also an extension in which we pick a set of coordinates and optimize them all at once. For example, we could optimize over an entire row of B at once rather than one entry at a time, because the columns are not coupled.

7. Topic Modeling

Suppose we want to find topics from which documents are drawn. How would we model that?

In the previous matrix factorization techniques, we imposed different constraints and parameters. How do we pick these parameters, e.g., the number of clusters, the number of speakers in an audio track, etc. We want to pick a small number of components, but enough such that we can decouple the signal well.

In topic modeling, the "number of topics" can also be hand-wavy. In many cases, people compare the output to expert-tagged datasets.

Notation 7.1. In the bag of words model, we define:

$$f_{td} \triangleq \frac{\text{number of times term } t \text{ appears in document } d}{\text{total number of words in document } d}$$

We assume this encodes the relative importance of that word in the document. Each document $x_d \in \mathbf{R}^T$ is a vector where each element $x_{d,t} = f_{td}$.

Remark. How does this compare to indicator variables? It can pick up on important words if they appear frequently. But sometimes a single word might only appear once but encapsulate the entire document (e.g., appear in the header).

Remark. How should we deal with documents that are extremely short? Another option is to take:

$$x_{d,t} = \varepsilon + (1-\varepsilon) f_{td}$$

with $\varepsilon = 0.1$. This is equivalent to placing a prior (from a statistical motivation).

How should we deal with words that appear in all documents?

Notation 7.2. We define, where t is the term, and D is the collection of documents:

$$IDF_{tD} = \begin{cases} 1 \\ \log\left(\frac{|D|}{\sum \mathbf{1}\{d:t \in D\}}\right) \\ \log\left(1 + \frac{|D|}{\sum \mathbf{1}\{d:t \in D\}}\right) \end{cases}$$

All three are possible choices.

Now, a possible choices of $x_{d,t}$ could be:

$$x_{d,t} = f_{td} \times IDF_{tD}$$

where the IDF weights the terms based on how often it is witnessed across documents. Thus, if a term is only seen in a subset of documents, we assume it is more informative in helping us discriminate topics.

For now, we consider the simplest version:

$$x_{d,t} = f_{td}$$

Therefore, we can construct the model:

$$x_d = Ab_d$$

where $A \in \mathbf{R}^{T \times K}$ is the topic matrix, $A \geq 0, \sum_{j=1}^{T} A_{ji} = 1$, and $b_d \in \mathbf{R}^K$, with $b_d \geq 0, \sum_i b_{di} = 1$.

Here, not only are the b_d constrained to live within the simplex, the columns of A are constrained to the simplex as well (because each column gives a multinomial over the terms for each document).

Consider the case where $b_d \in \{0, 1\}$, $\sum_{i=1}^{k} b_{di} = 1$. Then in this setting, b_d is an indicator vector giving the topic i to which x_d corresponds. That is;

$$x_d = Ab_d$$

selects a single column of A; so, in this setting,

$$x_d = A_{.i}$$

which is equivalent to saying that each document only has a single topic, so we expect the term frequencies in that document to correspond to the term frequencies for the topic to which it corresponds.

Example 7.3. Consider the following topics:

(1) windows, file, dos, files
(2) God, Jesus, bible, faith

Remark. Recall that in k-means, there were no constraints on A; the centers could be anywhere. Now, we constrain the column of A to be in the simplex.

But, the model where $x_d = Ab_d$ does not work well, because there is noise. So, we refine it such that we have:

$$x_d = Ab_d + w_d$$

where w_d is noise.

7.1. Solution via Alternating Minimization. We can now return to alternating minimization. Our goal is to find A, b_d; so we can try solving:

$$\hat{A} = \arg\min_{A \in \mathcal{A}} \sum_{d=1}^{n} \frac{1}{2} \|Ab_d - x_d\|_2^2$$

$$\hat{b}_d = \arg\min_{b_d \in \mathcal{B}} \sum_{d=1}^{n} \frac{1}{2} \|Ab_d - x_d\|_2^2$$

We can now rewrite this as a matrix factorization problem. Let us define $B = (b_1, b_2, \ldots, b_n)$ where $n = |D|$, the total number of documents. Observe that $B \in \mathbf{R}^{K \times n}$. That is, each column gives the topic loadings for each document (i.e., the proportion of each topic in each document).

We can thus write the problem for all documents together as:

$$\hat{A} = \arg\min_{A \in \mathcal{A}} \frac{1}{2} \|AB - X\|_F^2$$

$$\hat{B} = \arg\min_{B_{.d} \in \mathcal{B} \forall d} \frac{1}{2} \|AB - X\|_F^2$$

We again see that this is essentially soft-clustering, with the exception that the columns of A now must live in the simplex. When A was unconstrained, we could solve for it with least squares, and we turned to mirror descent for solving B.

7.1.1. *Algorithm.* `AltMin_TopicModels`

```
(1) while not finished:
    (a) $A^{k+1} = \arg\min_{A \in \mathcal{A}} \frac{1}{2} \|AB - X\|_F^2$
    (b) $B^{k+1} = \arg\min_{B_{.d} \in \mathcal{B}} \frac{1}{2} \|AB - X\|_F^2$
```

Observe that A, B are both matrices whose columns to constrained to live in the simplex. If we redefine:

$$\mathcal{A}(M, N) = \left\{ A \in \mathbf{R}^{M \times N} : A_{ij} \geq 0, \sum_{i=1}^{M} A_{ij} = 1 \right\}$$

then we can rewrite the algorithm as:

`AltMin_TopicModels`

```
(1) while not finished:
    (a) $A^{k+1} = \arg\min_{A \in \mathcal{A}(T,K)} \frac{1}{2} \|AB - X\|_F^2$
    (b) $B^{k+1} = \arg\min_{B \in \mathcal{A}(K,n)} \frac{1}{2} \|AB - X\|_F^2$
```

Note that we can use losses other than the Frobenius norm, for example the logistic loss. This will be addressed in the homework.

8. Non-Negative Matrix Factorization

NMF is similar to topic models. In its simplest form, we want to:

$$\begin{array}{ll} \underset{A,B}{\text{minimize}} & \frac{1}{2} \|AB - X\|_F^2 \\ \text{subject to} & A \in \mathbf{R}_+^{n \times k}, B \in \mathbf{R}_+^{k \times d}, \|B_{i.}\|_2^2 = 1 \end{array}$$

We may consider a more general version is considered:

$$\begin{array}{ll} \underset{A,B}{\text{minimize}} & \frac{1}{2} \|AB - X\|_F^2 + \lambda \left(\|A\|_1 + \|B\|_1 \right) + \mu \left(\|A\|_F^2 + \|B\|_F^2 \right) \\ \text{subject to} & A \in \mathbf{R}_+^{n \times k}, B \in \mathbf{R}_+^{k \times d} \end{array}$$

that is; the hard constraint is replaced with penalties, with an additional ℓ_1 penalty.

9. Independent Components Analysis

In Independent Components Analysis, we want to take signals and decompose them into "independent" parts. Let us begin with notation.

Notation 9.1. Let $X \in \mathbf{R}^{d \times n}$, with

$$X = AS$$

where $A \in \mathbf{R}^{d \times d}, S \in \mathbf{R}^{d \times n}$. We may interpret $S_{i.}$ as the i^{th} source.

We call A the "mixing matrix" and S the "source matrix".

Remark. A should be invertible. It's kind of cheating if we have one recording, and make it louder by a factor of two, and say it's another recording. We need each recording to contribute new information.

Example 9.2. Source separation between English and Spanish speaker from World Cup. Here, d is the number of "microphones", in this case, 2. n would be the length of the recording. For every source, we have a "mixed-up" audio recording.

Suppose we observe X. Our goal is to find:

$$S = A^{-1}X$$

without knowing A. We make two assumptions to make this problem tractable.

First, we want to impose structure on S such that the rows of S are independent. That is, the data are independent.

Second, we assume that the entries of S are *not* Gaussian-looking. Although the Gaussian assumption makes stuff nice to work with, it made this source separation problem more difficult. Empirically, if we took a histogram of all the points in a row of S, they don't look "bell-shaped".

There are some issues that we can't recover from:

(1) We can't distinguish between S and a row-permuted S'.
(2) We can't distinguish between scalings of the rows, i.e., S and a matrix S' where an arbitrary row is scaled by a factor of α.

But these two are not fatal; we can still recover the original audio signals and normalize. But if amplitude is important, we'll have to take on extra assumptions.

9.1. Classical ICA. In the normal ICA problem, we want to find S. The classical way to do this is to recover the rows of A^{-1} one at a time.

Let w_1^T be the first row of A^{-1}. This implies that:

$$\begin{aligned} w_1^T X &= w_1^T AS \\ &= \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix} S \\ &= S_{1.} \end{aligned}$$

We may thus find w_1 using the second assumption. We want to find the w_1 which makes $w_1^T X$ look the "least Gaussian". Note that $w_1^T X \in \mathbf{R}^n$. How can we make this as "not-bell-shaped" as possible. To avoid setting w_1 to zero, we take on a further assumption:

$$\frac{1}{1}\sum_{i=1}^{n} \left[(w_1^T X)_i - \bar{\mu}\right]^2 = 1$$

where $\bar{\mu} = \frac{1}{n}\sum_{i=1}^{n}(w_1^T X)_i$. That is; we want the sample variance to be not be zero. This brings us to the second issue of ICA. Because we cannot distinguish between scaling, we make the assumption that every row of S has sample variance 1.

Before ICA, statisticians, applied mathematicians, chemists, wanted to find projections of data that do not appear Gaussian. This is because if we hit random data X with projections, it tends to look Gaussian.[why?]We want to find some measure of "non-Gaussianity".

NOTATION 9.3. Let $\nu \sim N(\mathbf{0}_{M\times 1}, 1)$ be a sample from `np.random.randn(M)` where M is very large. Now, pick some function G that is not quadratic.

Under these assumptions, we will find that the solution:

$$w_1^* = \arg\max_{w_1} \left[\mathbf{E}[G(w_1^T X)] - \mathbf{E}[G(\nu)]\right]^2$$

yields $\operatorname{var}({w_1^*}^T X) = 1$, and $\operatorname{var}(\nu) \approx 1$ because of the distribution from which it is drawn.

REMARK. We do not want G to be quadratic because we already have a quadratic and so it will be bad.$^{\text{[citation needed]}}$

9.2. Solution for Classical ICA. We want to solve:

$$w_1^* = \arg\max_{w_1} \left[\mathbf{E}[G(w_1^T X)] - \mathbf{E}[G(\nu)]\right]^2$$

First, we examine a common choice of G:

$$G(s) = \frac{1}{a_1}\log\cosh(a_1 s)$$

where $1 \leq a_1 \leq 2$, $\cosh(t) = \frac{e^t + e^{-t}}{2}$. This is related to the cumulative distribution function.$^{\text{[why?]}}$

9.2.1. *Pre-Processing Steps for ICA.* Before we present the algorithm, we pre-process the data.

(1) Center the rows of the data matrix X. That is, $\sum_{j=1}^n X_{ij} = 0$ for all $i \in [d]$. Let $\mu = \frac{X\mathbf{1}_n}{n}$, we can center X by replacing it with $X - \mu\mathbf{1}^T$. We now assume that X is row-centered.
(2) Whiten X. We want to find a matrix M such that $\frac{MXX^TM^T}{n} = \mathbf{I}$. MX is a transformation of the columns such that the covariance of the data is the identity. This can be achieved via the singular value decomposition.
    Assume $X = UDV^T$. Then $M = UD^{-1}U^T$. We now assume without loss of generality that X is row-centered and whitened.

How does this translate to S? We make two assumptions on S:

(1) $\frac{1}{n}\sum_{j=1}^n S_{ij} = 0$; that is, the rows of S are centered.
(2) For all i, the $\operatorname{var}(S_{i.}) = 1$.

Using the "independence" assumption, we can easily show that

$$SS^T = n\mathbf{I} \qquad \text{(Recall that } S \in \mathbf{R}^{d\times n}.\text{)}$$

That is; all of the sources are uncorrelated. So, we have:

$$\begin{aligned} MXXM^T &= MASS^TA^TM^T \\ &= nMA\mathbf{I}A^TM^T \\ &= n(MA)(MA)^T \\ &= n\mathbf{I} \end{aligned}$$

Now, we let $X \triangleq MX$ and $A \triangleq MA$. Under $X\mathbf{1}_n = 0$, we have $w_1^T X\mathbf{1}_n = 0 \ \forall \ w_1$, which implies that

$$\begin{aligned} \operatorname{var}(w_1^T X) &= n \cdot w_1^T X X^T w_1 \\ &= \|w_1\|_2^2 \\ &= 1 \end{aligned}$$

which implies that $A^{-1} = A^T$ after transformation.

To continue, let us consider a few definitions:

Definition. For a vector $z \in \mathbf{R}^n$,

$$\mathbf{E}Z = \frac{1}{n}\sum_{i=1}^{n} Z_i$$

$$\operatorname{var}(Z) = \frac{1}{n}\left[Z_i - \mathbf{E}Z\right]^2$$

Recall that our first goal is to find a row of A^{-1} and call it w_1. To satisfy our assumptions for "identifiability", we want the projection of the data onto w_1 is non-Gaussian. We assume without loss of generality that $\operatorname{var}(w_1^T X) = 1$.

That is, we want to find w_1 that maximizes the "non-Gaussianity" of $w_1^T X$ subject to $\operatorname{var}(w_1^T X) = 1$.

Definition. Our measure of "non-Gaussianity" is

$$\left[\mathbf{E}(G(w_1^T X)) - G(\nu)\right]^2$$

where G is some function and ν is something that looks very Gaussian. For example, we can take $\nu \sim \mathcal{N}_M(0, \mathbf{I})$ for $M \gg n$.

So, one of the interpretations for Independent Components Analysis is solving the problem:

$$\begin{array}{ll} \underset{w_1}{\text{maximize}} & \left[\mathbf{E}(G(w_1^T X)) - G(\nu)\right]^2 \\ \text{subject to} & \operatorname{var}(w_1^T X) = 1 \end{array}$$

Why don't we want G to be quadratic? This is because the variance of G is already quadratic. But what if we had an adversary that knew G? Then it could cook up a non-Gaussian distribution ρ such that $G(\rho)$ does look Gaussian.

Two typical choices for fast Independent Components Analysis are:

$$G(s) = \frac{1}{a_1}\log\cosh(a, s) \qquad 1 \le a_1 \le 2$$

$$G(s) = -\exp\left\{-\frac{s^2}{2}\right\}$$

These are nice because they take into account every moment. The takeaway is that there can be a bad G. If it doesn't make sense, then try another G.

We assume that our data is centered and whitened. That is, we assume without loss of generality that:

$$X\mathbf{1} = 0$$
$$\frac{XX^T}{n} = \mathbf{I}_{k\times k}$$

due to the centering, whitening, zero mean, and unit variance assumptions.

This allows us to simply the objective.

$$\begin{aligned}\mathrm{var}(w_1^TX) &= \frac{1}{n}\sum_{i=1}^n \left((w_1^TX)_i - \mathbf{E}[w_1^TX]\right)^2\\ &= \frac{1}{n}\sum_{i=1}^n \left((w_1^TX)_i\right)^2\\ &= \frac{w_1^TXX^Tw_1}{n}\\ &= \|w_1\|_2^2\end{aligned}$$

Therefore, we may rewrite the optimization as:

$$\begin{aligned}&\underset{w_1}{\text{maximize}} && \left[\mathbf{E}(G(w_1^TX)) - G(\nu)\right]^2\\ &\text{subject to} && \|w_1\|_2^2 = 1\end{aligned}$$

This looks increasingly simliar to problems we have already solved. We are just maximizing over a vector. This looks like something we can solve via gradient ascent.

To optimize, we see that the maximum is achieved when $\mathbf{E}\left[G(w_1^TX)\right]$ is maximized or minimized. Finding that is hard, so instead, we try to find a stationary point. Gradient ascent is excellent for finding stationary points.

So, we have a couple options:

(1) Simply run projected gradient ascent on $\mathbf{E}\left[G(w_1^TX)\right]$ subject to $\|w\| = 1$.
(2) (Fast ICA): Basically uses gradient ascent, but with Newton's method.

9.3. Fast ICA. If we want to solve a non-linear system of equations, to find a stationary point, we can take the gradient $g(s) = G'(s)$. But because of the constraint, we must first introduce a Lagrange multiplier again. We now instead maximize over w_1 and minmiize over μ:

$$\frac{1}{n}\sum_{i=1}^n G(w_1^TX_{.i}) + \mu(\|w_1\|^2 - 1)$$

Now, we can take the gradient and set it to zero.

$$\nabla G = \frac{1}{n}\sum_{i=1}^n g({w_1^*}^TX_{.i})X_{.i} + 2\mu w_1^* = 0$$

Newton's method tells us to approximate this with a linear function, and set that equal to zero.

Suppose the current iterate is w_1^0. Then we approximate:

$$\nabla w_1 \approx \frac{1}{n}\sum_{i=1}^n g({w_1^0}^T X_{.i})X_{.i} + 2\mu w_1 + \underbrace{\frac{1}{n}\sum_{i=1}^n g'({w_1^0}^T X_{.i})X_{.i}X_{.i}^T}_{\text{Hessian}}(w_1 - w_1^0)$$

Therefore, the fast ICA approximation assumes:

$$\frac{1}{n}\sum_{i=1}^n g'({w_1^0}^T X_{.i})X_{.i}X_{.i}^T \approx \left(\frac{1}{n}\sum_{i=1}^n g'({w_1^0}^T X_{.i})\right)\mathbf{I}$$

which implies:

$$\frac{1}{n}\sum_{i=1}^n g({w_1^0}^T X_{.i})X_{.i} + 2\mu w_1 + \left(\frac{1}{n}\sum_{i=1}^n g'(w_1^T X_{.i})\right)(w_1 - w_1^0) = 0$$

Then, we have:

$$\begin{aligned}-(1+2\mu)w_1 &= \frac{1}{n}\sum_{i=1}^n g(w_1^0 X_{.i})X_{.i} - \frac{1}{n}\sum_{i=1}^n g'({w_1^0}^T X_{.i})w_1^0 \\ \Rightarrow w_1 &\propto \frac{1}{n}\sum_{i=1}^n g(w_1^0 X_{.i})X_{.i} - \frac{1}{n}\sum_{i=1}^n g'({w_1^0}^T X_{.i})w_1^0\end{aligned}$$

subject to the constraint that $\|w_1\|_2^2 = 1$

9.3.1. *Algorithm.* This yields the Fast ICA algorithm:

```
(1) while not finished:
    (a) $w_1^+ = \frac{1}{n}\sum_{i=1}^n g(w_1^T X_{.i})X_{.i} - \left(\frac{1}{n}\sum_{i=1}^n g'(w_1^T X_{.i})\right) w_1$
    (b) $w_1 = \frac{w_1^+}{\|w_1^+\|_2}$
```

9.4. Extensions. What about the rest of the sources?

9.4.1. *Projection pursuit.*

```
(1) For each $w_k$ in an ordered fashion:
    (a) while not finished:
        (i) Step (1a) of Fast ICA.
        (ii) $w_k^+ = w_k^+ - \sum_{j=1}^{k-1} w_j w_j^T w_k^+$; that is project onto the orthogonal complement of the previous $w_k$. (Gram-Schmidt.)
        (iii) $w_k = \frac{w_k^+}{\|w_k^+\|_2}$
```

9.4.2. *Parallel Computations.*

```
(1) while not finished:
    (a) for each $k$:
        (i) Step (1a) of Fast ICA.
```

(ii) $w_k = \frac{w_k^+}{\|w_k^+\|_2}$

(b) $(w_1, \dots, w_p) = \text{Gram-Schmidt}(w_1^+, \dots, w_p^+)$.

9.5. Maximum Likelihood.

$$\underset{\mathbf{w} = (w_1, \dots, w_p)}{\text{maximize}} \qquad \sum_{i=1}^{n} \left[\sum_{j=1}^{p} \log p_s(w_j^T X_{.i}) + \log\det \mathbf{w} \right]$$

where $p_s(t)$ is some measure.

10. Matrix Completion & the Missing Entry Problem

10.1. Motivation. Suppose we have a matrix $M \in \mathbf{R}^{d_1 \times d_2}$, where we observe some of the entries. Our goal is to fill in the missing entries based on what has been observed.

Example 10.1. Movie Recommendations, Collaborative Filtering, the Netflix Prize Problem

Suppose d_1 are users, and d_2 are movies. Each users have an underlying rating for each movie, but we only observe some of them. We may also have information about the users and the movies (e.g. demographic information about users, movie genres, etc.).

10.2. Setup. The most popular model is a low rank factorization, where $M = AB, A \in \mathbf{R}^{d_1 \times k}, B \in \mathbf{R}^{k \times d_2}$. The underlying assumption is that users and movies can be expresed in terms of features. We might think that we cannot observe this feature space, but we can embed users and movies in this k-dimensional feature space.

Suppose the observations are $\Omega = \{(i, j, M_{ij}) | M_{ij} \text{ was revealed}\}$.

10.3. Problem. We want to solve the problem:

$$\hat{A}, \hat{B} = \arg\min \sum_{(i,j)|M_{ij} \text{ revealed}} \left(M_{ij} - A_{i.} B_{j.}^T \right)^2$$

We can compare this to:

$$\hat{A}, \hat{B} = \arg\min \|M - AB\|_F^2$$

Many people add a ridge regression penalty to this formulation:

$$\hat{A}, \hat{B} = \arg\min \sum_{(i,j)|M_{ij} \text{ revealed}} \left(M_{ij} - A_{i.} B_{j.}^T \right)^2 + \lambda_A \|A\|_F^2 + \lambda_B \|X\|_F^2$$

which improves the statistical and computation performance. Using this model would have yielded a 7% improvement over Netflix's existing algorithms at the time.

10.4. Solution. This problem can be solved with alternating minimization. Random initalization typically works fine.

However, it works better if initialized from the singular value decomposition. To do this, we need to set the missing entries set to zero and center the matrix such that the average of the entries is zero.

There was some theory about this assuming the entries are revealed at random (i.e., movies are watched at random).

The other justification for using zeros (in this case, if the ratings are one through five) for missing entries is that people tend to watch movies that they like. So, if they haven't watched a movie, it takes a lot of evidence to overcome the belief that they don't like the movie.

10.5. Alternative Solutions. We can think of this as measuring similarities between users and movies.

We can also think that there are clusters of users, and run k-means on the users. Then we would have a sparse A; the communities to which a user belongs.

The addition that pushed the method over the top was incorporating temporal trends.

CHAPTER 3

Optimization

1. Stochastic Gradient Descent

1.1. Introduction. Many machine learning problems are of the form:

$$\arg\min_{w \in \mathbf{R}^d} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(x_i, y_i, w)$$

where $\mathcal{L}$ is the loss, $\{(x_i, y_i)\}_{i=1}^{n}$ is our data, and w are some parameters that we wish to learn.

Example 1.1. Squared loss.

$$\mathcal{L}(X, y, w) = \|Xw - y\|_2^2$$

1.2. Algorithm.

(1) while not finished:
    (a) Pick a random index $i \in [n]$.
    (b) $w_{t+1} = w_t - \eta_t \frac{1}{n} \nabla \mathcal{L}(x_i, y_i, w_t)$

Stochastic gradinet descent basically says, don't wait for all of the data to compute the full gradient, just update the weights as they stream in. This is well-adapted to large datasets.

1.3. Convergence. Although stochastic gradient descent will get you close to the solution quickly, its rate of convergence is only $\frac{1}{\sqrt{t}}$ or λ^t for $0 < \lambda < 1$, while the rate of gradient descent is like $\frac{n}{t}$, with the n on top because of the sum. We are ignoring the computation of the gradient here. We can see that eventually the rate of convergence for gradient descent is much faster.

This rate of convergence is for convex Lipschitz-smooth problems (regularity conditions). Note that the ℓ_1 norm does *not* satisfy this; this is why people invented ways to get around this, such as shrinkage.

2. Nestorov Acceleration

2.1. Introduction. Nesterov acceleration for both stochastic gradient descent converges with rate $\frac{n}{t^2}$ rate.

Remark. Nesterov acceleration can also be applied to vanilla gradient descent.

2.2. Algorithm. The update is given by:

$$w_t = (1+\mu)(w_{t-1} - \eta_t \nabla \mathcal{L}_i(w_{t-1})) - \mu(w_{t-2} - \eta_t \nabla \mathcal{L}_i(w_{t-2})$$

Observe that if $mu = 0$, we get normal gradient descent. This keeps track of the past motion, and tells it to go in the direction of where the current gradient wants you to go, but tempered by where the previous gradient wanted you to go.

This also have a momentum or velocity interpretation. An equivalent formulation is:

$$v_{t+1} = \mu v_t - \eta_t \nabla \mathcal{L}_i(w_t)$$
$$w_{t+1} = w_t - \eta_t \nabla \mathcal{L}_i(w_t) + \mu v_{t+1}$$

This interpretation is that one is moving with momentum proportional to μ, but gets 'bumped' in a new direction in each iteration (while maintaining some momentum).

There is some theory on picking μ, but only applies to convex optimization.

3. Picking Step Size

Oftentimes, people will run for a certain step size, wait until it stops improving, and then make η_t smaller.

We need to make η_t smaller because once in a while, we might get a bad example which could push us in the wrong direction. As we refine our running average, we don't want to let a single example push us off the current track.

Sometimes, people will run it for a while, and then keep η_t constant. The idea is that in stochastic gradient descent, each gradient is noisy. Eventually, we want some accuracy, so the η_t needs to be scaled to the accuracy that we want, and eventually we just keep that η_t.

4. Mini-Batch Stochastic Gradient Descent

4.1. Algorithm.

```
(1) while not finished:
    (a) $w_{t+1} = w_t \eta_t \sum_{i \in S_t} \nabla \mathcal{L}_i(w_i)$
```

This doesn't really help on a single computer because of the computational cost of the summations, but in a distributed setting, the summation can be performed in parallel.

5. Pseudo-Hessian Methods

This is an approximation to the full Newton Step in which rather than finding the full Hessian, we find a matrix which is easier to invert.

$$w_{t+1} = w_t - \eta H^{-1} \sum \nabla \mathcal{L}_i(w_t)$$

where H is a "good" matrix that approximates the Hessian.

Example 5.1. Limited-memory BFGS for normal gradient descent.

Example 5.2. AdaGrad and Adam.

In AdaGrad, the idea is that Hessian is a running sum of the squares of the past gradients. The intuition behind this is that very rarely seen stuff should have greater influence. It can also be interpreted as an adaptive step-size for each coordinate, but it essentially is a pseudo-Hessian method.

Adam is an improvement to AdaGrad which includes a dampening step and momentum.

6. Weight Decay

Weight decay is a slight adjustment to stochastic gradient descent.

6.1. Algorithm.

```
(1) while not finished:
    (a) $w_{t+1} = (1 - \eta_t \lambda) w_t - \eta_t \nabla \mathcal{L}_i(w_t)$
```

for a small λ. This is essentially ridge regression. Recall ridge regression:

$$\arg\min \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i(w) + \lambda \left\| w \right\|_2^2$$

we observe that its gradient is essentially the weight decay update.