Optimization & Linear Algebra

Idea: easily represent data mathematically. (With Linear Algebra)

Linear Algebra

Objective function of a model is to measure the distance between some properties of data, or some function of data. Linear algebra is the notation use to represent these measurements.

Measurements are with respect to the space they are in:

- **Scalars** are real numbers x $\in \Re$
- **Vectors** generalize scalars in *d* dimensions. $\mathbf{x} \in \Re^{\mathrm{d}}$
- **Matrices** generalize vectors in *m x n* dimensions. $\mathbf{x} \in \Re^{m\ x\ n}$
- **Tensors** generalize matrices in any dimensions $\mathbf{x} \in \Re^{a\ x\ b\ x\ c\ \cdots}$ (Representation of everything after matrix)

A **norm** is a function *f(x)* that assigns a strictly positive length or size to each vector in a vector space. The following qualities are important for distances. Norms qualities are…

- Non-negative ($f(\mathrm{x}) \geq 0$)
- Definite ($f(\mathrm{x}) = 0$ if and only if $\mathbf{x} = 0$)
- Homogenous ($f(t\mathrm{x}) = |t| f(\mathrm{x})$)
- Follows the triangle inequality ($f(\mathrm{x} + \mathrm{y}) \leq f(\mathrm{x}) + f(\mathrm{y})$)

The ℓ_p norm of a vector, $\mathbf{x}$, where $p \geq 1$, is denoted as,

$$||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}$$

The $\ell_{p,q}$ norm of a matrix, $\mathbf{X} \in \mathbb{R}^{n \times m}$, where $p, q \geq 1$, is denoted as,

$$||\mathbf{X}||_{p,q} = \left(\sum_{i=1}^{n} \left(\sum_{j=1}^{m} |x_{ij}|^p \right)^{\frac{q}{p}} \right)^{\frac{1}{q}} = \left(\sum_{i=1}^{n} ||\mathbf{x}^i||_p^q \right)^{\frac{1}{q}}$$

Class will focus mostly on l_1 and l_2 for p.

l_{pq} norm is used for matrices and has a nice relationships with traces. Class focus mostly on **Frobenius Norm** (p = 2, q = 2).

Matrix Properties and Operations

Symmetric Matrix

A symmetric matrix **S** is one where $\mathbf{s}_{ij} = \mathbf{s}_{ji}$.

Symmetric matrices have a special operation called **trace** which is the sum of the diagonal of that matrix represented as, $\mathbf{tr(S)} = \sum_{i=1}^{10} \mathbf{s}_{\mathrm{ij}}$

Rank

Rank of a matrix: maximum number of linearly independent column vectors or linearly independent row vector.

Max rank of a matrix is therefore less than the minimum of its dimensions.

$\mathbf{rank(x)} \leq \min(\mathrm{n,m})$ where $\mathbf{x} \in \mathrm{R}^{nxm}$

Low rank problem: happens in data, where you have lots of data/feature. This problem occur when feature starts depending on each other. So the problem is you not adding more feature because they all start to relate.

Transpose

Transpose of matrix is represented as $\mathbf{X}^{\mathbf{T}}$.

If $\mathbf{Y} = \mathbf{X}^{\mathbf{T}}$, then $y_{ij} = x_{ij}$.

If a matrix is a x b, transposing will make it b x a dimensions.

Symmetric matrices have a special property where $\mathbf{S}^{\mathbf{T}} = \mathbf{S}$

Inverse

Inverse $\mathbf{X}^{\text{-1}}$ of a matrix $\mathbf{X} \in \mathrm{R}^{nxn}$ is defined such that $\mathbf{A}^{\text{-1}}\mathbf{A} = \mathbf{A}\mathbf{A}^{\text{-1}}$

A matrix is **invertible** if its inverse exists and **singular** otherwise.

Eigenvalues / Eigenvectors

Eigenvalues λ and eigenvectors $\mathbf{v}$ of a matrix $\mathbf{X}$ satisfy, $\mathbf{X}\mathbf{v}_{\mathrm{i}} = \lambda_{\mathrm{i}}\mathbf{v}_{\mathrm{i}}$.

For a symmetric positive matrix $\mathbf{S}$, $\mathrm{eig}(\mathbf{S}^{\mathrm{T}}\mathbf{S}) = \mathrm{eig}(\mathbf{S}\mathbf{S}^{\mathrm{T}}) = \mathrm{eig}(\mathbf{S})$ o $\mathrm{eig}(\mathbf{S})$.

Orthogonal

Two vectors, $\mathbf{x} \in \mathrm{R}^n$, $\mathbf{y} \in \mathrm{R}^{in}$ are **orthogonal** if $\mathbf{x}^{\mathrm{T}}\mathbf{y} = 0$

A matrix $\mathbf{U}$, is orthogonal if every pair of its column vectors are orthogonal, $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$

Decomposition

Any matrix can be decomposed into the product of several matrices. If the matrix $\mathbf{X} \in \mathrm{R}^{nxm}$ is not symmetric we have,

Singular Value Decomposition (SVD): $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V^T}$

- $\mathbf{U} \in \mathrm{R}^{nxn}$ is the eigenvectors of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$
- $\mathbf{V} \in \mathrm{R}^{mxm}$ is the eigenvectors of $\mathbf{X}^{\mathrm{T}}\mathbf{X}$
- $\Sigma \in \mathrm{R}^{nxn}$ is a diagonal of the $\sqrt{eig(XX^T)}$

A special case of SVD exists when the matrix is symmetric…

Eigenvalue Decomposition: $\mathbf{X} = \mathbf{V}\Delta\mathbf{V}^{\mathrm{T}}$

- $\mathbf{V} \in \mathrm{R}^{\mathrm{nxn}}$ is the eigenvectors of $\mathbf{X}$
- Δ is a diagonal of the eigenvalues of $\mathbf{X}$

SVD is like factoring for matrixes

Eigenvalue decomposition is like SVD, but different results due to symmetric matrixes

Matrix Calculus

In conver optimization problems, you want to set derivative of a function to 0 to find a max or min.

Hessian Matrix is a square matrix of the second order partial derivatives of a function

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

Convex Optimization

A set is convex if every point in the set can create a line that lies within the function.

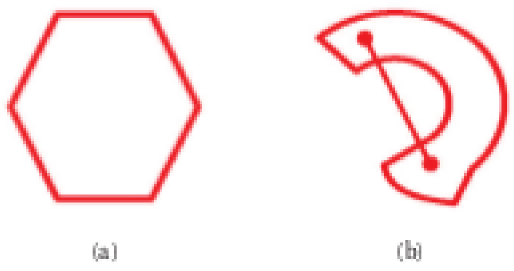

Figure 1: a is convex, b is not

Gradient Descent

Gradient Descent is a solution algorithm to optimizaion problem. Gradient descent focuses on **first order derivative** of the objective function. The idea behind it is

1. Select a valid point in the solution space
2. Calculate the graident at that point
3. Update the point by some amount by going down the gradient

Gradient is represented as ∇

Learning rate, or portion of the gradient to determine a step size is reprsented as γ

Gradient Descent is...

$$\mathbf{x}_{\mathrm{n}+1} = \mathbf{x}_{\mathrm{n}} - \gamma_{\mathrm{n}} \nabla f(\mathbf{x}_{\mathrm{n}})$$

Too large of a learning rate: overshoot optimal value Too small of a learning rate: take a long time to reach optimal

Newton's Method

Is like gradient descent but with additional information from second order derivative.

Newtons method is...

$$\mathbf{x}_{\mathrm{n}+1} = \mathbf{x}_{\mathrm{n}} - [\mathrm{H}(f(\mathbf{x}_{\mathrm{n}}))]^{-1} \nabla f(\mathbf{x}_{\mathrm{n}})$$

- $\mathrm{H}(f(\mathbf{x}_{\mathrm{n}}))$ is the Hessian matrix of f
- Can also add a learning rate to the step size for more granular control (similar to gradient descent)

Constrained Problems

So far the solutions presented were for *unconstrained* objective functions. To handle **constrained** problems we just convert it into unconstrained problems.

Lagrange multiplier, denoted as λ, is how we do this conversion.

Lagrangian function works by adding a variable to change in the optimization problem, the lagrange multiplier, and moving the constraint to the object function itself.

min f(x) => min f(x) + λg(x)

The second function is called the Lagrangian function.

Karush-Kuhn-Tucker (KKT)

To handle inequality constraints we follow the same approach to ensure KKT conditions. KKT condition states…

- g(x) ≤ 0
- $\lambda \leq 0$
- λg(x) ≤ 0