1 – Modern Hashing

Consistent Hashing and CMS

Consistent Hashing Very useful when need a hashtable distributed across multiple machines (e.g. shared cache). Hash all objects, but also hash all names of servers (to the same range), store objects in first server hashed to their right around the circle. Can reduce variance with virtual replication (with copies proportional to storage capacity). When new server is added, place it on the circle and rehash the things between it and the preceding bucket. To lookup the server, use balanced binary search tree to look up in $log(n)$ time, where n is the number of servers.

CMS There is no algorithm that exactly solves Heavy Hitters in one pass while using sublinear amount of auxiliar space. Approximate: require every value that occurs at least $\frac{n}{k}$ times is returned, but also that every value in the list occurs at least $\frac{n}{k} - \varepsilon n$ times in A (some false positives). CMS data structure has two operations, Inc(x) and Count(x). l x b table of l hash functions each with b buckets. For Count(x), return min value observed, for Inc(x) increment only set of minimum values. Bloom filter is CMS but with only binary entries - entry checking is "are ALL the buckets 1". Space required is bl counters where $b = \frac{e}{\varepsilon}$ and $l = ln\frac{1}{\delta}$, where δ is error probability. If n the length of the stream is not known in advance, keep track of running size of stream m and store elements in a min-heap that occur more than $\frac{m}{k}$ times. Prune minimum element if it falls below $\frac{m}{k}$. Heap contains at most $2k$ elements, so maintaining heap requires an extra $O(logk)$ work per array entry.

2 – Data with Distances

Jaccard $J(S,T) = \frac{|S \cap T|}{|S \cup T|}$.

$L_p \ \ ||x-y||_p = \left(\sum_{i=1}^{d} |x_i - y_i|^p\right)^{1/p}$

Voronoi diagram Given points X in $\mathbb{R}^d$, and some metric D, the Voronoi diagram partitions space into regions (the set of all points that are closest to a single datapoint).
Concretely, for $x \in X$,
$part(X) = eft\{y \in \mathbb{R}^d \text{ s. t. } D(x,y) = \min_{x' \in X} D(x',y).$
$k-d$ tree Given a set $S = \{x : x \in \mathbb{R}^d\}$. Pick a dimension / coordinate i. Compute the median of $\{x_i\}$. Partition: $S_1 = \{x \in S | x_i < m\}$, and $S_2 = \{x_i \geq m\}$. Recurse on S_1 and S_2 and store which dimension we are looking at. Runtime is logarithmic in number of points, and exponential in the dimension of points.

The kissing number. How many identical spheres can you place around a sphere such that all of them are touching the center sphere?

MinHash Consider Jacard similarity, defined earlier. We develop the "MinHash" technique that we can use to reduce dimesionality. Pick a uniformly random ordering of the universe U. Map set S to $f(S)$ = "min" element of S. For any sets S and T, we have $\Pr(f(S) = f(T)) = J(S,T)$.

MinHash Minhash preserves Jaccard similarities in expectation, like JL preserves l_2 in expectation. MinHash is an unbiased estimator of Jaccard Similarity. If we want an accurate estimate up to $\pm$ of all n choose 2 Jaccard similarities, use $O(\varepsilon^{-2} logn)$ independent estimates. Locality sensitive hashing is an extension that maps close things to close buckets (so you can search nearby buckets to remove near-duplicates, which is not possible with a hash).

KD Trees Perform well in <20 dimensions, or if number of points is at least 2^d. Build a binary search tree that partitions space; edges of tree correspond to subsets of space, and each node v has the index of some dimension i_v and a value m_v. Generally choose random dimension first and then choose m_v to be median of that dimension's coordinates. Given a node v, first go down the tree and find the leaf in which v would end up. Then go back up the tree, at each juncture, asking "Is it possible that the closest point to v would have ended up down the other path?". In low dimensions, answer is almost always no, but in high dimensions we end up exploring a lot (exponential in number of dimensions due to curse of dimensionality - hence bad performance).

JL transformation First, note that normals are closed. Let $X \sim (\mu_1, \sigma_1^2), Y \sim (\mu_2, \sigma_2^2)$. Then
$X + Y \sim (\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$
Note that this is fairly unique to Gaussians - this isn't true for most distributions. For any two vectors $x, y \in \mathbb{R}^d$, we have $\mathbf{E}[(f_r(x) - f_r(y))^2] = \mathbf{E}[||x-y||_2^2]$ Proof is easy:

$$\begin{aligned}
\mathbf{E}[(f_r(x) - f_r(y))^2] &= \mathbf{E}\left\{\left(\sum_{i=1}^{d} r_i x_i - \sum_{i=1}^{d} r_i y_i\right)^2\right\} \\
&= \mathbf{E}\left\{\left(\sum_{i=1}^{d} r_i (x_i - y_i)\right)^2\right\} \\
&= \mathbf{E}\left\{(0, \sum_i (x_i - y_i)^2)\right\} \\
&= \text{Var}\left\{(0, \sum_i (x_i - y_i)^2)\right\} \\
&= \sum_{i=1}^{n} (x_i - y_i)^2.
\end{aligned}$$

Note: If you repeat $l = \frac{\log n}{\varepsilon^2}$ times, then with high probability, all $\binom{n}{2}$ pairwise distances are preserved to within a factor of $1 \pm \varepsilon$.
This transformation is referred to as the Johnson-Lindenstrauss transformation.

JL Transform Gaussians are closed under addition! $X_1 + X_2$ has distribution $N(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. Variances add for **independent** random variables, expectations add for **any** variables. For a set of n points in k dimensions, to preserve all n choose 2 interpoint Euclidean distances up to a $1 \pm \varepsilon$ factor, set $d = O(\varepsilon^{-2} logn)$. JL transform sends x to $\frac{1}{d^{1/2}} Ax$ - don't forget scaling factor of $d^{\frac{-1}{2}}$! Preserves l_2 distances in reduced space. A is a d x k matrix. Can also just use ± 1 entries.

MinHash Minhash preserves Jaccard similarities in expectation, like JL preserves l_2 in expectation. MinHash is an unbiased estimator of Jaccard Similarity. If we want an accurate estimate up to $\pm$ of all n choose 2 Jaccard similarities, use $O(\varepsilon^{-2} logn)$ independent estimates. Locality sensitive hashing is an extension that maps close things to close buckets (so you can search nearby buckets to remove near-duplicates, which is not possible with a hash).

3 – Generalization

Consider the binary classification setting. We can broadly define it as follows: Suppose we have some datapoints $x_1, \ldots, x_n \in \mathbb{R}^d$. Known distribution D on $\mathbb{R}^d$. Ground truth label function $f : \mathbb{R}^d \to \{0,1\}$.

Problem. Given $x_1, \ldots, x_n$ drawn independently from D, and labels $f(x_1), \ldots, f(x_n)$, our goal is to output $g : \mathbb{R}^d \to \{0,1\}$ such that "$g \approx f$".
Namely, we want the generalization error to be low, defined this way: generalization error$(g) = \Pr_{x \sim D}[g(x) \neq f(x)]$. Can also define the training error as the fraction of the training points on which g disagrees with the true labelling.

Claim For any function g, the expected training error is equal to the generalization error.

Question. Suppose we find g with training error 0. When does this imply that the generalization error is small? Factors that influence this question: Amount of data (how faithful is the sample?). The complexity of the function (# of functions considered). Algorithm used to find g. This is often succinctly phrased as "does g generalize?" If answer is no, this implies that you've "overfit" the data. First, we'll consider the "well-separated finite setting." Here, we will make two enormous assumptions. Assume that: The ground truth labelling function $f \in S = \{f_1, f_2, \ldots f_k\}$. That is, f belongs to a set of functions with k elements which is finite. All of these functions are well separated. No function in the class is similar to f. For all $f_i \in S$ with $f_i \neq f$, the generalization error of $f_i >$. Note that this is sort of a silly assumption, but we will drop both.
Naive "algorithm:" return any g in our set S that has training error 0.

Main theorem Given assumptions 1 and 2, if the number of datapoints $n > \frac{1}{\varepsilon}(\log k + \log \frac{1}{\delta})$, then, with probability at least $1 - \delta$, g will generalize.
Some comments: logarithmic function in k and $1/\delta$ is good, but inverse linear function in $\frac{1}{\varepsilon}$ is kind of bad.
We will prove this in two parts. First, we will analyze the probability that we are "tricked" by a bad f_i. Next, we will union bound over all bad f_i's.
Consider a bad function f_i. The probability that we are tricked by this function is

$$\begin{aligned}
\Pr\{\text{TrainingError}(f_i) = 0\} &= \prod_{j=1}^{n} \Pr_{x_j \sim D}(f_i(x_j) = f(x_j)) \\
&\leq (1-\varepsilon)^n \\
&< e^{-n\varepsilon}.
\end{aligned}$$

The last inequality follows from the inequality $1 + x < e^x$.
Proof from Taylor series / plot.
There are at most k "bad" functions. Hence we can apply a union bound, to obtain
$\delta = \Pr(\text{output bad function}) \leq ke^{-n\varepsilon}$. Now, the desired result directly follows from solving for failure probability $\frac{1}{\delta}$.
Results of this form are generally referred to as being in the "PAC" framework (probably approximately correct).

Theorem for linear classifiers For linear classifiers, if the number of datapoints $n > \frac{C}{\varepsilon}(d + \log \frac{1}{\delta})$, then, with probability $1 - \delta$, g will generalize.

Generalization Does g generalize is equivalent to asking if its test error is the same as its training error. Start with two assumptions: 1) there are h (finite) possible ground truth functions, and every other candidate has generalization at least ε. Then probability of being tricked by a single function that matches on all n datapoints is $\leq (1-\varepsilon)^n \leq e^{-\varepsilon n}$. Union bounding over h possible functions, get probability $\leq he^{-\varepsilon n}$. Letting $n \geq \frac{1}{\varepsilon}(ln(h) + ln(\frac{1}{\delta}))$ then with probaility at least $1 - \delta$ we get the correct ground truth functions (sample complexity). Even if assumption (2) doesn't hold, can rephrase theorem as $n \geq \frac{1}{\varepsilon}(ln(h) + ln(\frac{1}{\delta}))$ implies with probability at least $1 - \delta$ over the samples, the output of the learning algorithm is a function with generalization error less than ε (PAC guarantee).
Can use curse of dimensionality for arbitrary linear classifiers - only exponentially many directions to look in (exp in dimension). Then if f is a linear classifier, $n \geq \frac{c}{\varepsilon}(d + ln(\frac{1}{\delta}))$, then with probability at least $1 - \delta$ the output of the learning algorithm is a function with generalization error less than ε. Rule of thumb: make sure you training data size n is linear in the number d of free parameters/features you're trying to learn.
If no function has 0 training error, output the function which has the smallest training error (ERM algorithm). Then if $n \geq \frac{c}{\varepsilon^2}(d + ln(\frac{1}{\delta}))$, with probability at least $1 - \delta$, the generalization error of the output is the training error $\pm\varepsilon$. Functions do not have to be linear classifiers, can be anything. In the same setting, if τ is the minimum generalization error of any linear classifier and $n \geq \frac{c}{\varepsilon^2}(d + ln(\frac{1}{\delta}))$, then the output of the ERM algorithm will have generalization error $\leq \tau + 2\varepsilon$. Note: from going to 0 percent error to nonzero error (i.e. realizable to non-realizable, we pick up another factor of $\frac{1}{\varepsilon}$.

Regularization Kernelization allows you to do polynomial embedding without actually writing out extra terms (do it implicitly instead). If we have a ton of data ($n > d$), then can use polynomial embeddings or random project + nonlinearity (neural networks). If the features in the input space are meaningful (i.e. coordinates matter), probably want polynomial embedding (or if you want to preserve interpretibility of features). If your features are rotationlliy invariatn, use random project + nonlinearity.
Bayesian viewpoint: true model underlying the data is drawn from a known prior. Assuming gaussian noise, finding max liklihood function is equivalent to L-2 regularization. It's unclear if Bayesian priors will hold - and note that different priors give rise to different regularizers. Frequentists, in contrast, argue that regularization allows you to recover properties of the true model (e.g. sparsity). l_1 regularizer is a great proxy for l_0 regularizer: Given n indepdent Gaussian vectors from R^d, consider labels $y_i =< a, x_i >$ for some vector a with $||a||_0 = s$, then the minimizer of the l_1 regularized objective function will be the vector a with high probability, provided that $n > cslog(d)$ for some constant c. Note the contrast to $O(d)$ data required with no regularization.

3 – Regularization

Note on last part - it is very open ended, at the cutting edge of machine learning research (but don't feel obliged to write pages of analysis).

Punchline from last class. If you a have a set k different functions $\{f_1, \ldots, f_k\}$, then the "best" one will generalize if $n > O(\log k)$. If we are classifying in d dimensions - we can approximate by set of $\exp(d)$ linear functions. If $n > d$, expect generalization.

Regularization. A way to express a set of preferences over models. Such a scheme that will take both performance on training data as well as these preferences into account. For example, we can consider L_2-regularized least squares. Let $x_1, \ldots, x_n \in \mathbb{R}^d$, and $y_1, \ldots, y_n \in \mathbb{R}$. In this setting, we want to minimize the following objective:
$\min_a f(a) = \sum_{i=1}^{n} (\langle x_i, a \rangle - y_i)^2 + \underbrace{\lambda ||a||_2^2}_{\text{regularization term}}$.
There are two broad types of regularization, explicit or implicit: Explicit regularization (e.g. L_2 regularization, preferring sparse vectors). Implicit regularization (algorithm itself has "preferences").
Why should we regularize; why wouldn't we just return the empirical risk minimizer?

Perspective. You always want roughly $n \approx d$, where generalization might not hold. If you have $n = 1,000,000$, then you want $d \approx 1,000,000$. Otherwise - it's sort of a "waste"; if d is truly 1000 in your dataset, try to construct additional features to learn a model in 1,000,000 dimensional space.
How should we construct additional features? Here are two approaches.

Polynomial embedding. (For example - quadratic embedding. $x = (x_1, x_2, \ldots x_d) \to f(x) =$
$(x_1, \ldots, x_d, x_1^2, x_1 x_2, x_1 x_3, \ldots, x_d^2, 1) \in \mathbb{R}^{2d+1+\binom{d}{2}}$ One simple setting in which you need quadratic features to fit a classifier is when you are fitting a circular decision boundary.

Random projection + non-linearity. You really need the non-linearity, since otherwise you'll just be learning another linear function. Can choose $\sqrt{x}, x^2, \sigma(x)$, or most other "nice" nonlinearities (since they all roughly have similar properties).
Downsides of adding new features: One objection could be that working with $\approx d^2$ dimensional points is annoying. But this isn't an issue: you can "implicitly" work over the embedded points without computing the embedding. The area of math devoted to this is referred to as "kernelization" (usually covered in the context of SVMs).
Real issue: if you need d^2 features, you generally need much more data.
Rule of thumb: if the coordinates actually have signifiance, the polynomial embedding preserves interpretability.
How should you think about regularization? There are two views: the Bayesian view, and the frequentist view.

Bayesian view. Assume that the true model is drawn from some known "prior" distribution. This allows us to evaluate the "likelihood" / probability of a given candidate model.

Frequentist view. Goal: argue that if true model has "nice structure," then can find it.

Bayesian approach to regularization.
("Gaussian prior") Suppose we have $x_1, \ldots, x_n \in \mathbb{R}^d$, assume that the true label $a^* \in \mathbb{R}^d$ is drawn by choosing each coordinate independently from $(0,1)$.
Now, suppose that each label is set as $y_i = \langle x_i, a^* \rangle + z_i$, where noise $z_i \sim (0, \sigma^2)$. Now, given $x_1, \ldots, x_n, y_1, \ldots, y_n$, ask Likelihood$(a) = \Pr(a)\Pr(\text{data}|a)$. Because we made strong assumptions on the label distributions, we can directly compute these probabilities. Hence

$$\begin{aligned}
\text{Likelihood}(a) &= \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi}} \exp(-a_i^2/2) \prod_{i=1}^{n} \exp(-(\langle x_i, a \rangle - y_i)^2/2\sigma^2) \\
&\propto \exp(-||a||_2^2/2 - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (\langle x_i, a \rangle - y_i)^2)
\end{aligned}$$

Maximum likelihood of a is equivalent to minimizing $\sum_{i=1}^{n} (\langle x_i, a \rangle - y_i)^2 + 2\sigma^2 ||a||_2^2$.
This derivation even told us how to set the regularization constant. Should be 2 times the variance of the noise.

Frequentist approach to sparsity / regularization. Consider a model a^* that is s-sparse (i.e. there are s nonzero coordinates).
Question: why do we care about sparse models?
One answer is that lots of models are actually sparse. Think of the laws of physics. Other view: maybe the world is sloppy, and the best model might not be sparse. But what's most helpful for interpretability is fitting a sparse model.
Question: can we build a regularizer that lets us selectively find sparse models? The obvious choice is
$f(a) = \sum_{i=1}^{n} (\langle x_i, a \rangle - y_i)^2 + \lambda ||a||_0$ where we are using the "0-norm" that computes sparsity.

Claim. If $n > O(s \log(d))$ then the sparsest model that fits the data is "correct."
The number of s-sparse d-dimensional function is just $\binom{d}{s}$, and there are about $\exp(s)$ sparse functions. So there are approximately $d^s \exp(s)$ sparse functions, and we need datapoints around the logarithm of this.
The problem with using sparse models is that it is not differentiable (so finding the minimizer is NP-hard).
So, we can note the following: l_0 regularization is great, but computationally intractable. Idea: use l_1 regularization as proxy for l_0.
Miraculously, the claim from before still holds for l_1 regularization (proved in the early 2000's, Candes in the stat / math department here).

4 – PCA

Principal component analysis can be used to analyze the structure of a data set or allow the representation of the data in a lower dimensional dataset (as well as many other applications).
Let $\{\vec{x}_i\}$ be a set of N column vectors of dimension D.
Define the scatter matrix S_x of the data set as
$S_x = \sum_{i=1}^{N} (\vec{x}_i - \vec{\mu}_x)(\vec{x}_i - \vec{\mu}_x)^T$ where $\vec{\mu}_x$ is the mean of the dataset $\vec{\mu}_x = \frac{1}{N} \sum_{i=1}^{N} \vec{x}_i$
The d largest principal components are the eigenvectors $\vec{w}_i$ corresponding to the d largest eigenvalues. d can be chosen arbitrarily with $d < D$. The eigenvectors of $\mathbf{S}$ can usually be found by using singular value decomposition.
The dominant eigenvectors describe the main directions of variation of the data. For example, if a dataset had 2 large eigenvalues, then the data variation is described largely by linear combinations of the 2 corresponding eigenvectors (ie. the data is largely coplanar).
The d eigenvectors can also be used to project the data into a d dimensional space. Define $\mathbf{W} = [\vec{\mu}_1, \vec{\mu}_2, \ldots, \vec{\mu}_d]$ The projection of vector $\vec{x}$ is $\vec{y} = \mathbf{W}^T \vec{x}$. The corresponding scatter matrix S_y of the vectors $\{\vec{y}_i\}$ is: $S_y = \mathbf{W}^T S_x \mathbf{W}$ The matrix $\mathbf{W}$ maximizes the determinant of S_y for a given d.

PCA PCA does not necessarily preserve pairwise distances like JL. JL is also data-oblivious, PCA is deterministic. Coordinates used in JL have no intrinsic meaning. PCA often finds latent variables. Linear regression has a "preferred" variable. PCA treats all coordinates equally - works well when there are a set of latent variables and all coordinates are linear combinations of those variables. PCA requires mean subtraction, and scaling of coordinates so that each coordinate has unit standard deviation. PCA assumes variance in the data corresponds to interesting information. PCA tries to maximize the square projection of each vector onto subspace, alternatively lower its square distances from the hyperplane. PCA does not find nonlinear structure. If the structure is non-orthogonal, PC components may lose interpretibility bc they're forced to be orthogonal. PCA can also fail due to high-variance noise. Can show with some extra work that we're trying to diagonalize the covariance matrix $X^T X$. Every covariance matrix can be diagonalized by an orthogonal matrix, i.e. $X^T X = QDQ^T$, Q orthogonal. If only need the top few principal components, compute them with power iteration: choose random unit vector, apply covariance matrix, rescale to unit, continue until convergence. Converged vector will be top principal component. Subtract out this PC from everything and reiterate. With probability at least $\frac{1}{2}$ over the choice of initial unit vector, for $t >= 1$, m datapoints in n dimensions, $|< A^t u_0, v1 >| \geq 1 - 2n\frac{1}{2}(\frac{\lambda_2}{\lambda_1})^t)$, , so number of iterations required scales as $\frac{logn}{log(\frac{\lambda_1}{\lambda_2})}$ (i.e. in the spectral gap). Can exponentially improve this probability $\frac{1}{2}$ by repeating this with different random initial unit vectors.

SVD Low-rank matrix approximations are great for matrix completions. $A = USV^T$. Left singular vectors are in U, right singular vectors are in V (remember diagram). U, V are orthogonal. S is diagonal matrix with entries sorted in nonincreasing order. Only for covariance matrix is $U = V$. The columns of U are called hte left singular vectors of A and the rows of V^T (columns of V are the right singular vectors of A). Running time is min $m^2 n$ or $n^2 m$. If just want k largest singular values and singular vectors, can do it in kmn time. Rank-k SVD is optimal in Frobenius norm of all rank-k matrices. How to choose k? Choose such that top k singular values sum to a multiple of the rest (maybe 10). The right singular vectors of X are the same as the eigenvectors of $X^T X$ (the principal components, and the eigenvalues of the covariance matrix are the square of the singular values of X. So PCA reduces to computing the SVD of A without forming the covariance matrix. If you only need the top fiew eigenvectors, use Power Iterations If you want them all, use SVD. In frobenius norm, each row of $A - A_k$ is exactly the distance from each poitn to the span of the top k principal components. SVD and PCA are equivalent. The rows of X can be interpreted as linear combinations of the rows of V^T. The columns of X can be interpreted as linear combinations of the columns of U (the left singular vectors). When the colunsn of U are interpretable, this is valuable. Otherwise, use PCA. In customers x products, the right singular vectors are customer types and left singular vectors are product types. Can develop PCA (not SVD) based low-rank approximations by computing covariance, and each row as the projection onto the top k principapl components, which is exactly the same as rank-k SVD approximation.

5 – SVD

Rank A matrix has rank k if it can be written as the sum of k rank one matrices.

Low rank approximation examples Compression, de-noising, matrix-completion.

SVD A SVD of an $m \times n$ matrix A can be written as $A = USV^T$, where U is $m \times m$ ortho, V is $n \times n$ ortho, and S is an $m \times n$ diagonal matrix with nonnegative entries (diagonal entries sorted from high to low). Columns of U: left SVs, cols of V: right SVs, and entries of S are singular values.
Note that SVD is equivalent to $A = \sum_{i=1}^{\min(m,n)} s_i \cdot u_i v_i^T$.
Runtime is smaller of $O(n^2 m)$ and $O(m^2 n)$. If you just want the largest k SVs, can compute this significantly faster, in time $O(kmn)$.

Rank k approx. Rank k approximation is $A_k = U_k S_k V_k^T$. Storing the matrices in rank k is $O(k(m+n))$, compared to $O(mn)$ space for original matrix A. Note that $||A - A_k||_F \leq ||A - B||_F$, for every matrix A, rank target $k \geq 1$, and rank k $m \times n$ matrix B.

PCA reduces to SVD We start by sketching the equivalence of truncated rank k SVD and a PCA projection onto the first k principal components. Suppose $\mathbf{X}$ is an $n \times p$ matrix (assume it is mean normalized, for simplicity). The $p \times p$ covariance matrix is computed as $\mathbf{X}^\mathbf{T}\mathbf{X}$. Since it is symmetric, we can diagonalize it as $\mathbf{X}^\mathbf{T}\mathbf{X} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\mathbf{T}$ where $\mathbf{Q}$ is a matrix of eigenvectors, and Λ is a diagonal matrix with eigenvalues λ_i in decreasing order on the diagonal. Projecting our data onto these eigenvectors yield the principal components.
Now, performing singular value decomposition gives us $\mathbf{X} = \mathbf{USV}^\mathbf{T}$, where $\mathbf{U}$ is a unitary matrix, and $\mathbf{S}$ is the diagonal matrix of singular values s_i. Expressing the covariance matrix in terms of the singular value decomposition of $\mathbf{X}$, we can write
$\mathbf{X}^\mathbf{T}\mathbf{X} = \mathbf{VSU}^\mathbf{T}\mathbf{USV}^\mathbf{T}/(n-1) = \mathbf{V}\frac{\mathbf{S}^2}{n-1}\mathbf{V}^T$. Therefore, the right singular vectors $\mathbf{V}$ are the eigenvectors of $\mathbf{X}^\mathbf{T}\mathbf{X}$. Furthermore, singular values can be expressed in terms of the eigenvalues of $\mathbf{X}^\mathbf{T}\mathbf{X}$ with the relation $\lambda_i = s_i^2/(n-1)$.

5B – Tensors

Tensors Examples: 3-Tensor with works that appear next to each other, moments tensor. n-Tensor's basis are outer product of n different sets of vectors. FACTS: For tensors, cannot iterative ssubtract rank-1 aproximation and get best rank $k-1$ approximation. Also do not get the rank over the reals is the same as the rank over complex numbers for k-tensors with $k \geq 3$, even if tensor is real-valued. Also don't know how to construct n x n x n tensors whose rank is greater than $n^{1.1}$ for all n - hard to reason about rank of tensors. Computing the rank of 3-tensors is NP-hard. But, if the rank of a 3-tensor is sufficiently small, its rank can be efficiently computed and its low-rank representation is unique and can be efficiently recovered.
Thm: Given a 3-tensor of rank k, the rank k decomposition is unique up to scaling vectors by a constant and these factors can be efficiently recovered. The factors do NOT need to be orthogonal, as long as they are linearly independent.
Skipping proof.

6 – Spectral Graph Theory

Graph Given a graph $G = (V, E)$ with $|V| = n$ vertices, can associate matrices to graph. Laplacian matrix is $L_G = D - A$, where D is degree matrix (diagonal matrix where $D(i,i)$ is degree of ith node in G), and A is the adjacency matrix, with $A(i,j) = 1$ iff $(i,j) = E$. Note that $v^t Lv = \sum_{i<j:(i,j) \in E} (v(i) - v(j))^2$. v assigns number of each vertex in G, and $v^t Lv$ is sum of squares of differences between values of neighboring nodes.

Eigenspaces From the previous identity, eigenvalues are nonnegative. Number of zero eigenvalues of L is the number of connecting components of G. The maximum eigenvalue $\max_{||v||=1} v^t Lv$ will maximize the discpreancy between neighbors values of v.

Spectral embeddings Smallest eigenvector is not helpful, since $v_1 = (\frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}})$. Helpful to consider v_2 and v_3, and plot these together.

Isoperimetric ratio Given a set S, the isoperimetric ratio $\theta(S)$ is defined as $\theta(S) = \frac{|\delta(S)|}{\min(|S|, |V \setminus S|)}$. The isoperimetric number is $\theta_G = \min_{S \subset V} \theta(S)$.

Conductance The conductance of a partition of a graph is defined as $cond(S) = \frac{|\delta(S)|}{\min(\sum_{i \in S} deg(i), \sum_{i \notin S} deg(i))}$.

λ_2 Isoperimetric number of graph satisfies $\theta_G \geq \lambda_2 (1 - \frac{|S|}{|V|})$.

Partial converse (Cheger) If λ_2 is second smallest eigenvalue, there exists a set $S \subset V$ such that $cond(S) \leq \frac{\sqrt{2\lambda_2}}{\sqrt{d}}$.

7 – Sampling / Estimation

RS - problem Want to sample k uniformly random elements from a datastream of length $N \gg k$. N large, and N unknown.

Reservoir sampling Given a number k, and a datastream $x_1, x_2, \ldots$ of length $> k$: Put the first k elements of the stream into a reservoir $R = (x_1, \ldots, x_k)$. For $i \geq k+1$, withj probability $\frac{k}{i}$ replace a random entry of R with x_i. Return reservoir at the end of stream.

RS is uniform For $t \geq i$, $\Pr[x_i \in R_t] = \frac{k}{t}$ where R_t is the reservoir after time t. Proof by induction.

Markov Let X be a real-valued RV with $X \geq 0$. For any $c > 0$, $\Pr[X \geq c\mathbf{E}[X]] \leq \frac{1}{c}$. Intuition: at most 10% of population can have income $> 10 \times$ avg. income.

Chebyshev Let X be real-valued RV, and $c > 0$. Then $\Pr\left[|X - \mathbf{E}[X]| \geq c\sqrt{\text{Var}[X]}\right] \leq \frac{1}{c^2}$.

Importance sampling We can estimate properties of distribution A, based on samples from distribution B. Idea: have distribuyion B place higher weight on the "important" elements of the domain.

Importance sampling ex. For example, suppose we know that computer scientists have higher income, and that computer scientists are 0.05 of population. Instead of taking n random samples from pop, take $0.8n$ samples of non-computer scientists, and 0.2 samples of random computer sciences. Then: $avg = 0.95a_1 + 0.05a_2$ allows us to estimate the population average salary. Benefit: reduce variance of the estimate, by focusiong our samples on the important portion of the distribution.

Good-Turing frequency estimation Given n independent draws from a distribution, estimate: $\Pr[\text{next draw new}] \approx \frac{\#\text{ elts seen once}}{n}$.

Monte Carlo method Broadly, learning about a distribution via random sampling. MCMC: Markov Chain Monte Carlo - applying Monte Carlo to Markov Processes.

Markov Process Has two parts: a set of states $S = \{s_1, s_2, \ldots\}$ and a transition function $P : S \times S \to [0,1]$, where $P(s_i, s_j)$ is the probability that the process transitions $s_i \to s_j$. For any s_i, $\sum_j P(s_i, s_j) = 1$. Markov property: distribution of X_i depends only on X_{i-1}.

Calculating $D(t,s)$ Let $D(t,s)$ be the distribution over values of X_t. Can directly calculate $D(t,s) = D(0,s) \cdot P^t$. Requires $O(\log t)$ matrix multipilcations using the "repeated squaring" trick. Alternatively, can use Monte Carlo. Start with $X_0 = s$, and for all $i = 1, \ldots, t$ simulate the selection of X_i aaccording to P and X_{i-1}. If you do this k times, have obtained k samples drawn from the distribution $D(t,s)$. Time to compute is $kt \cdot updateT$, where $updateT$ is the time to sample from the distribution defined by a row of P.

Fundamental Theorem Suppose 1) $\forall$ pairs s_i, s_j, we can reach s_j if we start at s_i. 2) Chain is aperiodic, i.e. $\gcd(\{t_1, \ldots\}) = 1$, where $\{t_1, \ldots\}$ is the set of times consisting of all t for which $\Pr[X_t = s_j | X_0 = s_i] > 0$. Basically: as long as the chain is not a directed cycle or a bipartite graph. Then, for all states s, $\lim_{t \to \infty} D(t,s) \to \pi$.

Mixing time "Time until we are close to π" is the mixing time of the distriobution. Formally,
$mixingTime = \min\{t : \max_s ||D(t,s) - \pi||_{TV} \leq \frac{1}{4}\}$.
That is, the minimum time t s.t. no matter where we start, the distance between $D(t,s)$ and π is at most $\frac{1}{4}$.

Eigenvalues / Power iteration Difficult to estimate the mixing time of a given chain. Ratio of second largest eigenvalue to largest, $\frac{\lambda_{max-1}}{\lambda_{max}}$ determines the rate aat which chain will converge to π.

8 – Fourier Analysis

Transform Given a length n vector, the Fourier transform $\mathscr{F}(v)$ is a complex valued vector of length n defined as $\mathscr{F}(v) = M_n v$, where j,k entry of M_n is defined to be w_n^{jk}, where $w_n = \exp(-\frac{2\pi i}{n})$.

Transform 2 Consider a sequence of complex numbers $\{x_i\}$. The FT is $X_k = \sum_{n=0}^{N-1} x_n \cdot \exp(-\frac{2\pi i}{N} kn)$.

Properties FFT is $O(n \log n)$. Inverse Fourier transform is multiplication by matrix M^{-1} whose j,k entry is w^{-jk}. FT is almost its own inverse.

Fourier basis Can interpret v as a linear combination in the Fourier basis $v = \sum_{j=0}^{n} c_j b_j$. The j-th coordinate in the Fourier transform Mv tells you how much of the j-th freqwuency the signal v has.

FT as polynomial ops Given a length n vector v, define $P_v(x) = v_0 + v_1 x + \cdots + v_{n-1} x^{n-1}$. The k-th entry of the transform Mv is $v_0 + v_1 w^k + \cdots + v_{n-1} w^{(n-1)k}$. Inverse transform takes a vector representing P_v evaluated as roots of unity, and returns coefficients of polynomial P_v (interpolation).

FFT Given a length $n = 2^k$ vector v, we output its Fourier transform. Recursively compute $q_1 = \mathscr{F}(w_{even})$ and $q_2 = \mathscr{F}(v_{odd})$. Define the length $n/2$ vector s whose j-th index is $\exp(-2\pi ij/n)$. Output the concatenation of $q_1 + (q_2 \times s)$ and $q_1 - (q_2 \times s)$, where $q \times s$ is elementwise product. This decomposition only involves computation of two Fourier transforms of length $n/2$ vectors, and two element-wise multiplications and additions. $T(n)$ satisfies $T(n) = 2T(n/2) + O(n)$.

Convolution Convolution of two vectors v, w of lengths n, m is the vector of coefficients in the product of the polynomials associated with v, w. We can write $v * w = \mathscr{F}^{-1}(\mathscr{F}(v) \times \mathscr{F}(w))$, where $\times$ is element-wise product.

Properties of convolutions Liunearity. $(v+w)*a = (v*a) + (w*a)$ and $(cv)*a = c(v*a)$. Translational invariance. Shifting a vector by padding with k zeros and cconvolving is same as shifting the convolution.

Why convolutions are important Any transformation of a vector that is linear and translation invariant is a convolution. Can use this fact to simulate physical systems, e.g. gravity.

9 – Compressive Sensing

Problem setup Design m "linear measurements" $a_1, \ldots, a_m \in \mathbb{R}^n$. Nature picks a "signal" $z \in \mathbb{R}^n$. Compute $b_i = \langle a_i, z \rangle$. Use the vector b to recovere z.

Idea The idea of compressive sensing is to capture data in a compressed form (i.e. dot product with certain vectors). Applications: lower power cameras, MRI and tomography, demystifying undersampling (don't need as many samples as we thought - e.g. in oil drilling). Goal is to design m linear measurements in R^n that can recover any unknown signal z in R^n. Get a vector b in R^m of the dot products, want to recover z from b by solving linear program. Can we get away with $m < n$ measurements? Not unless z is sparse (otherwise z is underdetermined). NOTE: in images, usually black is 0 and white is 1. Usually assume z is sparse in the standard basis, but can also focus on sparsity in Fourier basis or other bases and throw an extra change-of-basis matrix into some of the statements below.

Main theorem Fix signal length n and a sparsity level k. Let A be an $m \times n$ matrix of iid standard Gaussian RVs with $m = O(k \log \frac{n}{k})$ rows. For every k-sparse signal z, the unique solution to $\min ||x||_1$ subject to $Ax = b$ is z.

Theorem comments Extends to "almost k-sparse" signals. Extends to non Gaussian matrices A. However, sparse matrices A don't work. m is optimal up to a constant factor, since $\log_2 \binom{n}{k} = O(k \log \frac{n}{k})$ by Stirling's approximation.

Uncertainty principle No signal can be made sparse simultaneously in both time and frequency domains.

LPs Consist of decision variables, linear constraints of the form $\sum_{j=1}^n a_{ij}x_j \leq b_i$, or $\sum_{j=1}^n a_{ij}x_j = b_i$. Objective function should be linear, of the form $\min \sum_{j=1}^n c_j x_j$.

ℓ_1 **linearization** Add y_j to represent $|x_j|$. Use objective function $\min \sum_{j=1}^n y_j$ which is linear, with $2n$ inequalities of the form $y_j - x_j \geq 0, y_j + x_j \geq 0$. Can also extend LPs to accommodate noise.

Convexity A set $C \subseteq \mathbb{R}^n$ is convex for every $x, y \in C$ and $\lambda \in [0,1]$, $\lambda x + (1-\lambda)y \in C$. A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if the region above the graph is a convex set. That is, $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$ for every $x, y \in \mathbb{R}^n$ and $\lambda \in [0,1]$. Note that LPs are convex, since intersection of half-spaces will be convex. Also, the set of $n \times n$ symmetric and PSD matrices viewed as a subset of $\mathbb{R}^{n^2}$ is convex.

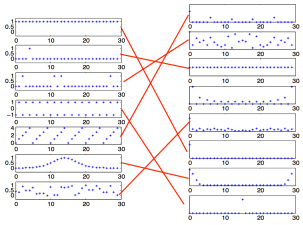
Matrix completion Suppose we have a matrix $\hat{M}$ derived from M by erasing its entries. Want to recover M, with key assumption that M has low rank. Suppose M has SVD USV^T. Rank of M is the number of non-zero signular values, with every row of M a linear combination of top r right SVs, and every col is a linear combination of top r left SVs. Optimization problem can be viewed as $\min |\text{supp}(\Sigma(M))|$, subject to M agrees with $\hat{M}$ on known entries. The exact problem is ℓ_0, but we can relax it to optimization the ℓ_1 norm with $\min ||\Sigma(M)||_1$ (which is convex).

Matrix completion theorem If M has rank r, and $\hat{M}$ has at least $\Omega(r(m+n)\log^2(m+n))$ known entries, chosen uniformly at random from M. Then M is sufficiently dense and non-pathological.

Application: Matrix Completion Key assumption: we want to recover the entries of a partially observed matrix M (observed is $Mhat$) that **HAS LOW RANK**. If we try to minimize the rank of M subject to M agreeing with $Mhat$ on observed entries, we get NP-hard problem. But if we reframe, we can say we want to minimize the number of nonzero singular values of M s.t. M agrees with $Mhat$ on known entries. Can again change this to **nuclear norm minimization** where we want to minimize the sum of the singular values and solve: $min||S(M)||_1$ s.t. M agrees with $Mhat$ on known entries. In fact, the objective function here is convex and can be solved efficiently.

Assignments

Let X, Y be discrete integer random variables. Then we can write the distribution of $Z = X + Y$ as a convolution between X and Y: $P(Z = z) = \sum_{k=-\infty}^{\infty} P(X = k)P(Y = z - k)$. In particular, the vector $q = p * p * \cdots * p$ (convolved 100 times) represents the probability distribution over 100 rolls of a six sided die. Since $q[0]$ represents the probability that we roll a total of 100, $q[150]$ will represent the probability that the 100 rolls sum to 250.



Let $\mathbf{f}, \mathbf{g}$ be N-tuples. Let $\mathbf{h} = \mathbf{f} * \mathbf{g}$. Computing the Fourier transform, we obtain

$$\begin{aligned}
\mathbf{F_h}(m) &= \sum_{j=0}^{2N-1} h[j] \exp(-2\pi i m j/N) \\
&= \sum_{j=0}^{2N-1} \left(\sum_{k=0}^{2N-1} f[k] g[j-k] \right) \exp(-2\pi i m j/N) \\
&= \sum_{k=0}^{2N-1} f[k] \left(\sum_{j=0}^{2N-1} g[j-k] \exp(-2\pi i m j/N) \right) \\
&= \sum_{k=0}^{2N-1} f[k] \left(\exp(-2\pi i m k/N) \mathscr{F}(g^+) \right) \\
&= \mathscr{F}(g^+) \sum_{k=0}^{2N-1} f[k] (\exp(-2\pi i m k/N)) \\
&= \mathscr{F}(f^+) . \mathscr{F}(g^+),
\end{aligned}$$

which proves the desired result.
Therefore, we can implementation the convolution $\mathbf{f} * \mathbf{g}$ in the frequency domain as follows: $\mathbf{f} * \mathbf{g} = ^{-1} (\mathbf{f}^+ \cdot \mathbf{g}^+)$. A naive implementation of convolution would require $O(N^2)$ operations. Since the complexity of the FFT is $O(n \log n)$, a FFT-based convolution is also $O(n \log n)$, and therefore much more efficient.
If $\mathbf{f}$ is an N-tuple and $\mathbf{g}$ is an M-tuple with $M < N$ we can zero-pad $\mathbf{f}$ and $\mathbf{g}$ to length $M + N$, and perform the same computation.

PCA does poorly with noise only in Y because it aims to capture the variance in the y direction because of noise. As the noise coefficient c increases, PCA will upweight the principal component in the y direction, which increases the slope as c increases.
Formally, the first principal component satisfies

$$\begin{aligned}
\mathbf{w}_1 &= _{||\mathbf{w}||=1} \left\{ \sum_i (\mathbf{x}_i \cdot \mathbf{w})^2 \right\} \\
&= _{||\mathbf{w}||=1} \left\{ \sum_i ((i/1000, 2i/1000 + \mathscr{N}(0,c)) \cdot \mathbf{w})^2 \right\}.
\end{aligned}$$

Since the noise is biased towards the second component of x, the net effect is to scale each dot product, biasing the first principal component towards the y direction.

PCA does well with noise in both X and Y, because there is now variance in both dimensions. In this setting, we can compute $\mathbf{w}_1$ as follows:

$$\begin{aligned}
\mathbf{w}_1 &= _{||\mathbf{w}||=1} \left\{ \sum_i (\mathbf{x}_i \cdot \mathbf{w})^2 \right\} \\
&= _{||\mathbf{w}||=1} \left\{ \sum_i ((i/1000 + \mathscr{N}(0,c), 2i/1000 + \mathscr{N}(0,c)) \cdot \mathbf{w})^2 \right\}.
\end{aligned}$$

Since the variance is spread across both dimensions, $\mathbf{w}$ does not get "biased" towards either dimension, and PCA recovery results in a good estimate of the slope.

Least squares performs poorly because with increasing noise in X and Y, there is less of a predictive relationship between X and Y.
The least squares slope estimate can be computed as follows:

$$\langle X_n - \overline{X_n}, Y_n - \overline{Y_n} \rangle / / ||X_n - \overline{X_n}||_2^2,$$

where

$$X_n = X + \mathscr{N}(0,c)$$
$$Y_n = Y + \mathscr{N}(0,c).$$

This results in a decreasing slope as the noise coefficient c increases. In particular, as $c \to \infty$, the slope $\to 0$, since there will be no correlation between X and Y.

New stuff

Frequentists vs. Bayesians what is probability?
One is called the **frequentist** interpretation. In this view, probabilities represent long run frequencies of events. For example, the above statement means that, if we flip the coin many times, we expect it to land heads about half the time. The other interpretation is called the **Bayesian** interpretation of probability. In this view, probability is used to quantify our **uncertainty** about something; hence it is fundamentally related to information rather than repeated trials (Jaynes 2003). In the Bayesian view, the above statement means we believe the coin is equally likely to land heads or tails on the next toss
One big advantage of the Bayesian interpretation is that it can be used to model our uncertainty about events that do not have long term frequencies. For example, we might want to compute the probability that the polar ice cap will melt by 2020 CE. This event will happen zero or one times, but cannot happen repeatedly. Nevertheless, we ought to be able to quantify our uncertainty about this event. To give another machine learning oriented example, we might have observed a "blip" on our radar screen, and want to compute the probability distribution over the location of the corresponding target (be it a bird, plane, or missile). In all these cases, the idea of repeated trials does not make sense, but the Bayesian interpretation is valid and indeed quite natural. We shall therefore adopt the Bayesian interpretation in this book. Fortunately, the basic rules of probability theory are the same, no matter which interpretation is adopted.

A brief review of probability theory

Basic concepts We denote a random event by defining a **random variable** X.
Descrete random variable: X can take on any value from a finite or countably infinite set.
Continuous random variable: the value of X is real-valued.
CDF

$$F(x) \triangleq P(X \leq x) = \begin{cases} \sum_{u \leq x} p(u) & \text{, discrete} \\ \int_{-\infty}^{x} f(u)\mathrm{d}u & \text{, continuous} \end{cases} \quad (1)$$

PMF and PDF For descrete random variable, We denote the probability of the event that $X = x$ by $P(X = x)$, or just $p(x)$ for short. Here $p(x)$ is called a **probability mass function** or **PMF**.A probability mass function is a function that gives the probability that a discrete random variable is exactly equal to some value[1]. This satisfies the properties $0 \leq p(x) \leq 1$ and $\sum_{x \in \mathscr{X}} p(x) = 1$.
For continuous variable, in the equation $F(x) = \int_{-\infty}^{x} f(u)\mathrm{d}u$, the function $f(x)$ is called a **probability density function** or **PDF**. A probability density function is a function that describes the relative likelihood for this random variable to take on a given value[2].This satisfies the properties $f(x) \geq 0$ and $\int_{-\infty}^{\infty} f(x)\mathrm{d}x = 1$.

Mutivariate random variables
Joint CDF We denote joint CDF by $F(x,y) \triangleq P(X \leq x \cap Y \leq y) = P(X \leq x, Y \leq y)$.

$$F(x,y) \triangleq P(X \leq x, Y \leq y) = \begin{cases} \sum_{u \leq x, v \leq y} p(u,v) \\ \int_{-\infty}^{x} \int_{-\infty}^{y} f(u,v)\mathrm{d}u\mathrm{d}v \end{cases} \quad (2)$$

product rule:

$$p(X,Y) = P(X|Y)P(Y) \quad (3)$$

Chain rule:

$$p(X_{1:N}) = p(X_1)p(X_2|X_1)p(X_3|X_2,X_1)...p(X_N|X_{1:N-1}) \quad (4)$$

Marginal distribution **Marginal CDF**:

$$\begin{cases} F_X(x) \triangleq F(x, +\infty) = \\ \sum_{x_i \leq x} P(X = x_i) = \sum_{x_i \leq x} \sum_{j=1}^{+\infty} P(X = x_i, Y = y_j) \\ \int_{-\infty}^{x} f_X(u)du = \int_{-\infty}^{x} \int_{-\infty}^{+\infty} f(u,v)\mathrm{d}u\mathrm{d}v \end{cases} \quad (5)$$

$$\begin{cases} F_Y(y) \triangleq F(+\infty, y) = \\ \sum_{y_j \leq y} p(Y = y_j) = \sum_{i=1}^{+\infty} \sum_{y_j \leq y} P(X = x_i, Y = y_j) \\ \int_{-\infty}^{y} f_Y(v)dv = \int_{-\infty}^{+\infty} \int_{-\infty}^{y} f(u,v)\mathrm{d}u\mathrm{d}v \end{cases} \quad (6)$$

Marginal PMF and PDF:

$$\begin{cases} P(X = x_i) = \sum_{j=1}^{+\infty} P(X = x_i, Y = y_j) & \text{, descrete} \\ f_X(x) = \int_{-\infty}^{+\infty} f(x,y)\mathrm{d}y & \text{, continuous} \end{cases} \quad (7)$$

$$\begin{cases} p(Y = y_j) = \sum_{i=1}^{+\infty} P(X = x_i, Y = y_j) & \text{, descrete} \\ f_Y(y) = \int_{-\infty}^{+\infty} f(x,y)\mathrm{d}x & \text{, continuous} \end{cases} \quad (8)$$

Conditional distribution **Conditional PMF**:

$$p(X = x_i|Y = y_j) = \frac{p(X = x_i, Y = y_j)}{p(Y = y_j)} \text{ if } p(Y) > 0 \quad (9)$$

The pmf $p(X|Y)$ is called **conditional probability**.
Conditional PDF:

$$f_{X|Y}(x|y) = \frac{f(x,y)}{f_Y(y)} \quad (10)$$

Bayes rule

$$\begin{aligned} p(Y = y|X = x) &= \frac{p(X = x, Y = y)}{p(X = x)} \\ &= \frac{p(X = x|Y = y)p(Y = y)}{\sum_{y'} p(X = x|Y = y')p(Y = y')} \end{aligned} \quad (11)$$

Independence and conditional independence We say X and Y are unconditionally independent or marginally independent, denoted $X \perp Y$, if we can represent the joint as the product of the two marginals, i.e.,

$$X \perp Y = P(X,Y) = P(X)P(Y) \quad (12)$$

We say X and Y are conditionally independent(CI) given Z if the conditional joint can be written as a product of conditional marginals:

$$X \perp Y|Z = P(X,Y|Z) = P(X|Z)P(Y|Z) \quad (13)$$

Quantiles Since the cdf F is a monotonically increasing function, it has an inverse; let us denote this by F^{-1}. If F is the cdf of X, then $F^{-1}(\alpha)$ is the value of x_α such that $P(X \leq x_\alpha) = \alpha$; this is called the α quantile of F. The value $F^{-1}(0.5)$ is the **median** of the distribution, with half of the probability mass on the left, and half on the right. The values $F^{-1}(0.25)$ and $F^1(0.75)$are the lower and upper **quartiles**.

Mean and variance The most familiar property of a distribution is its **mean**,or **expected value**, denoted by μ. For discrete rv's, it is defined as $\mathbb{E}[X] \triangleq \sum_{x \in \mathscr{X}} xp(x)$, and for continuous rv's, it is defined as $\mathbb{E}[X] \triangleq \int_{\mathscr{X}} xp(x)\mathrm{d}x$. If this integral is not finite, the mean is not defined (we will see some examples of this later).
The **variance** is a measure of the "spread" of a distribution, denoted by σ^2. This is defined as follows:

$$\begin{aligned} var[X] &= \mathbb{E}[(X - \mu)^2] \\ &= \int (x - \mu)^2 p(x)\mathrm{d}x \\ &= \int x^2 p(x)\mathrm{d}x + \mu^2 \int p(x)\mathrm{d}x - 2\mu \int xp(x)\mathrm{d}x \\ &= \mathbb{E}[X^2] - \mu^2 \end{aligned} \quad (14)$$

from which we derive the useful result

$$\mathbb{E}[X^2] = \sigma^2 + \mu^2 \quad (15)$$

The **standard deviation** is defined as

$$std[X] \triangleq \sqrt{var[X]} \quad (16)$$

This is useful since it has the same units as X itself.
Some common discrete distributions In this section, we review some commonly used parametric distributions defined on discrete state spaces, both finite and countably infinite.
The Bernoulli and binomial distributions Now suppose we toss a coin only once. Let $X \in \{0,1\}$ be a binary random variable, with probability of "success" or "heads" of θ. We say that X has a **Bernoulli distribution**. This is written as $X \sim \text{Ber}(\theta)$, where the pmf is defined as

$$\text{Ber}(x|\theta) \triangleq \theta^{\mathbb{I}(x=1)}(1-\theta)^{\mathbb{I}(x=0)} \quad (17)$$

Suppose we toss a coin n times. Let $X \in \{0,1,\cdots,n\}$ be the number of heads. If the probability of heads is θ, then we say X has a **binomial distribution**, written as $X \sim \text{Bin}(n,\theta)$. The pmf is given by

$$\text{Bin}(k|n,\theta) \triangleq \binom{n}{k} \theta^k (1-\theta)^{n-k} \quad (18)$$

The multinoulli and multinomial distributions The Bernoulli distribution can be used to model the outcome of one coin tosses. To model the outcome of tossing a K-sided dice, let $\vec{x} = (\mathbb{I}(x=1), \cdots, \mathbb{I}(x=K)) \in \{0,1\}^K$ be a random vector(this is called **dummy encoding** or **one-hot encoding**), then we say X has a **multinoulli distribution**(or **categorical distribution**), written as $X \sim \text{Cat}(\theta)$. The pmf is given by:

$$p(\vec{x}) \triangleq \prod_{k=1}^{K} \theta_k^{\mathbb{I}(x_k=1)} \quad (19)$$

Suppose we toss a K-sided dice n times. Let $\vec{x} = (x_1, x_2, \cdots, x_K) \in \{0,1,\cdots,n\}^K$ be a random vector, where x_j is the number of times side j of the dice occurs, then we say X has a **multinomial distribution**, written as $X \sim \text{Mu}(n,\vec{\theta})$. The pmf is given by

$$p(\vec{x}) \triangleq \binom{n}{x_1 \cdots x_k} \prod_{k=1}^{K} \theta_k^{x_k} \quad (20)$$

where $\binom{n}{x_1 \cdots x_k} \triangleq \frac{n!}{x_1!x_2!\cdots x_K!}$
Bernoulli distribution is just a special case of a Binomial distribution with $n = 1$, and so is multinoulli distribution as to multinomial distribution. See Table **??** for a summary.

The Poisson distribution We say that $X \in \{0,1,2,\cdots\}$ has a **Poisson distribution** with parameter $\lambda > 0$, written as $X \sim \text{Poi}(\lambda)$, if its pmf is

$$p(x|\lambda) = e^{-\lambda} \frac{\lambda^x}{x!} \quad (21)$$

The first term is just the normalization constant, required to ensure the distribution sums to 1.
The Poisson distribution is often used as a model for counts of rare events like radioactive decay and traffic accidents.
The empirical distribution The **empirical distribution function**[3], or **empirical cdf**, is the cumulative distribution function associated with the empirical measure of the sample. Let $\mathscr{D} = \{x_1, x_2, \cdots, x_N\}$ be a sample set, it is defined as

$$F_n(x) \triangleq \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(x_i \leq x) \quad (22)$$

Some common continuous distributions In this section we present some commonly used univariate (one-dimensional) continuous probability distributions.
Gaussian (normal) distribution If $X \sim N(0,1)$,we say X follows a **standard normal** distribution.
The Gaussian distribution is the most widely used distribution in statistics. There are several reasons for this.

1. First, it has two parameters which are easy to interpret, and which capture some of the most basic properties of a distribution, namely its mean and variance.
2. Second,the central limit theorem (Section TODO) tells us that sums of independent random variables have an approximately Gaussian distribution, making it a good choice for modeling residual errors or "noise".
3. Third, the Gaussian distribution makes the least number of assumptions (has maximum entropy), subject to the constraint of having a specified mean and variance, as we show in Section TODO; this makes it a good default choice in many cases.
4. Finally, it has a simple mathematical form, which results in easy to implement, but often highly effective, methods, as we will see.

See (Jaynes 2003, ch 7) for a more extensive discussion of why Gaussians are so widely used.

If $\nu = 1$, this distribution is known as the **Cauchy** or **Lorentz** distribution. This is notable for having such heavy tails that the integral that defines the mean does not converge.
To ensure finite variance, we require $\nu > 2$. It is common to use $\nu = 4$, which gives good performance in a range of problems (Lange et al. 1989). For $\nu \gg 5$, the Student distribution rapidly approaches a Gaussian distribution and loses its robustness properties.
Here μ is a location parameter and $b > 0$ is a scale parameter. See Figure **??** for a plot.
Its robustness to outliers is illustrated in Figure **??**. It also put mores probability density at 0 than the Gaussian. This property is a useful way to encourage sparsity in a model, as we will see in Section TODO.
Introduction Linear regression is the "work horse" of statistics and (supervised) machine learning. When augmented with kernels or other forms of basis function expansion, it can model also nonlinear relationships. And when the Gaussian output is replaced with a Bernoulli or multinoulli distribution, it can be used for classification, as we will see below. So it pays to study this model in detail.
Representation

$$p(y|\vec{x}, \vec{\theta}) = \mathscr{N}(y|\vec{w}^T\vec{x}, \sigma^2) \quad (23)$$

where $\vec{w}$ and $\vec{x}$ are extended vectors, $\vec{x} = (1,x)$, $\vec{w} = (b,w)$. Linear regression can be made to model non-linear relationships by replacing $\vec{x}$ with some non-linear function of the inputs, $\phi(\vec{x})$

$$p(y|\vec{x}, \vec{\theta}) = \mathscr{N}(y|\vec{w}^T\phi(\vec{x}), \sigma^2) \quad (24)$$

This is known as **basis function expansion**. (Note that the model is still linear in the parameters $\vec{w}$, so it is still called linear regression; the importance of this will become clear below.) A simple example are polynomial basis functions, where the model has the form

$$\phi(x) = (1, x, \cdots, x^d) \quad (25)$$

MLE Instead of maximizing the log-likelihood, we can equivalently minimize the **negative log likelihood** or **NLL**:

$$\text{NLL}(\vec{\theta}) \triangleq -\ell(\vec{\theta}) = -\log(\mathscr{D}|\vec{\theta}) \quad (26)$$

The NLL formulation is sometimes more convenient, since many optimization software packages are designed to find the minima of functions, rather than maxima.
Now let us apply the method of MLE to the linear regression setting. Inserting the definition of the Gaussian into the above, we find that the log likelihood is given by

$$\ell(\vec{\theta}) = \sum_{i=1}^{N} \log \left[\frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{1}{2\sigma^2} (y_i - \vec{w}^T\vec{x}_i)^2 \right) \right] \quad (27)$$

$$= -\frac{1}{2\sigma^2} \text{RSS}(\vec{w}) - \frac{N}{2} \log(2\pi\sigma^2) \quad (28)$$

RSS stands for **residual sum of squares** and is defined by

$$\text{RSS}(\vec{w}) \triangleq \sum_{i=1}^{N} (y_i - \vec{w}^T\vec{x}_i)^2 \quad (29)$$

We see that the MLE for $\vec{w}$ is the one that minimizes the RSS, so this method is known as **least squares**.
Let's drop constants wrt $\vec{w}$ and NLL can be written as

$$\text{NLL}(\vec{w}) = \frac{1}{2} \sum_{i=1}^{N} (y_i - \vec{w}^T\vec{x}_i)^2 \quad (30)$$

There two ways to minimize $\text{NLL}(\vec{w})$.

OLS Define $\vec{y} = (y_1, y_2, \cdots, y_N)$, $\vec{X} = \begin{pmatrix} \vec{x}_1^T \\ \vec{x}_2^T \\ . \\ . \\ \vec{x}_N^T \end{pmatrix}$, then $\text{NLL}(\vec{w})$ can be written as

$$\text{NLL}(\vec{w}) = \frac{1}{2} (\vec{y} - \vec{X}\vec{w})^T (\vec{y} - \vec{X}\vec{w}) \quad (31)$$

When $\mathscr{D}$ is small(for example, $N < 1000$), we can use the following equation to compute directly

$$\hat{w}_{\text{OLS}} = (\vec{X}^T\vec{X})^{-1}\vec{X}^T\vec{y} \quad (32)$$

The corresponding solution $\hat{w}_{\text{OLS}}$ to this linear system of equations is called the **ordinary least squares** or **OLS** solution.
We now state without proof some facts of matrix derivatives (we won't need all of these at this section).

$$\begin{aligned} trA &\triangleq \sum_{i=1}^{n} A_{ii} \\ \frac{\partial}{\partial A} AB &= B^T \quad (33) \\ \frac{\partial}{\partial A^T} f(A) &= \left[\frac{\partial}{\partial A} f(A) \right]^T \quad (34) \\ \frac{\partial}{\partial A} ABA^TC &= CAB + C^TAB^T \quad (35) \\ \frac{\partial}{\partial A} |A| &= |A|(A^{-1})^T \quad (36) \end{aligned}$$

Then,

$$\begin{aligned} \text{NLL}(\vec{w}) &= \frac{1}{2N} (\vec{X}\vec{w} - \vec{y})^T (\vec{X}\vec{w} - \vec{y}) \\ \frac{\partial \text{NLL}}{\vec{w}} &= \frac{1}{2} \frac{\partial}{\vec{w}} (\vec{w}^T\vec{X}^T\vec{X}\vec{w} - \vec{w}^T\vec{X}^T\vec{y} - \vec{y}^T\vec{X}\vec{w} + \vec{y}^T\vec{y}) \\ &= \frac{1}{2} \frac{\partial}{\vec{w}} (\vec{w}^T\vec{X}^T\vec{X}\vec{w} - \vec{w}^T\vec{X}^T\vec{y} - \vec{y}^T\vec{X}\vec{w}) \\ &= \frac{1}{2} \frac{\partial}{\vec{w}} tr(\vec{w}^T\vec{X}^T\vec{X}\vec{w} - \vec{w}^T\vec{X}^T\vec{y} - \vec{y}^T\vec{X}\vec{w}) \\ &= \frac{1}{2} \frac{\partial}{\vec{w}} (tr\vec{w}^T\vec{X}^T\vec{X}\vec{w} - 2tr\vec{y}^T\vec{X}\vec{w}) \end{aligned}$$

Combining Equations 36 and 37, we find that

$$\frac{\partial}{\partial A^T} ABA^TC = B^TA^TC^T + BA^TC$$

Let $A^T = \vec{w}, B = B^T = \vec{X}^T\vec{X}$, and $C = I$, Hence,

$$\begin{aligned} \frac{\partial \text{NLL}}{\vec{w}} &= \frac{1}{2} (\vec{X}^T\vec{X}\vec{w} + \vec{X}^T\vec{X}\vec{w} - 2\vec{X}^T\vec{y}) \\ &= \frac{1}{2} (\vec{X}^T\vec{X}\vec{w} - \vec{X}^T\vec{y}) \\ \frac{\partial \text{NLL}}{\vec{w}} &= 0 \Rightarrow \vec{X}^T\vec{X}\vec{w} - \vec{X}^T\vec{y} = 0 \\ \vec{X}^T\vec{X}\vec{w} &= \vec{X}^T\vec{y} \quad (37) \\ \hat{w}_{\text{OLS}} &= (\vec{X}^T\vec{X})^{-1}\vec{X}^T\vec{y} \end{aligned}$$

Equation 39 is known as the **normal equation**.

Final Eigenvalues of Laplacian matrix of the complete graph: if eigs are $\lambda_1, \ldots, \lambda_n$, then eigenvalues of Laplacian are $d - \lambda_1, \ldots, d - \lambda_n$.

Why better for Count to return min instead of average? a) Since we always increment CMS[i][hi(x)] for each i when x appears, CMS[i][hi(x)] can only be an overestimate of the actual count of x. Therefore taking the minimum over all i provides a closer estimate of the count of x than taking the average.

PCA vs JL a) PCA computes the k (k«m) eigenvectors of the covariance matrix of m data points, which correspond to the largest eigenvalues, trying to express the data points as a sum of appropriately scaled vectors, thus revealing low-dimensional linear structure in the data. JL multiplies each of the input data point with an appropriate matrix of i.i.d. entries from a standard Gaussian distribution thus producing a data point with fewer coordinates (the coordinates now have no intrinsic meaning) as the original. However, this mapping has the property that pairwise euclidean distances are preserved. JL has also the nice property of being data-oblivious. b) PCA is more appropriate for visualization. Usually plotting the first 2 or 3 components (as we did with genome data) will contain a lot of interesting insights about the data. JL reduces the dimension but not to a point where visualization is intuitive. c) PCA exactly tries to capture the low-dimensional linear structure as it tries to express all data points as the sum of a few vectors scaled appropriately. However, if the structure is low-dimensional but non-linear PCA will fail. d) JL because if the noise is in a particular direction then the relative distances wouldn't be influenced a lot. PCA as we saw in the problem set tries to capture this noise as part of a latent variables' dependence and doesn't perform well. e) JL would be appropriate for NN search queries and classification since it approximately preserves the pairwise (euclidean) distances used in the computations of NN. PCA offers no guarantees about preserving pairwise distances.

[1] http://en.wikipedia.org/wiki/Probability_mass_function
[2] http://en.wikipedia.org/wiki/Probability_density_function
[3] http://en.wikipedia.org/wiki/Empirical_distribution_function