CS 181 LECTURE NOTES

AARON LANDESMAN

Contents

1. Class 1/29/14

1.1. **K-means clustering.**

(1) Where are the cluster centers? $\{\underline{\mu_k}\}_{k=1}^K, \underline{\mu}_k \in \mathbb{R}^D$. with D the same as for the date.
(2) Which data belong to which cluster?

Definition 1.1.1. *A **one-hot coding** is a map $s \mapsto e^{i^T}$, the transpose of a basis vector*

Assign responsibility vectors $\underline{r}_n$ given by one-hot codings. If the data belongs to cluster k, then define

$$r_{n,j} = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{otherwise} \end{cases}$$

To measure distances, might typically use euclidean norm

$$||\underline{x} - \underline{\mu}||^2 = \sum_{i=1}^{n} (x_i - \mu_i)^2.$$

1.1.2. *Loss Function.*

Remark 1.1.3. *A loss functions is a way of formalizing badness.*

Distortion for data might be $J_n(\underline{r}_n, \{\underline{\mu}_k\}_{k=1}^K) = \sum_{k=1}^K r_{n,k} ||\underline{x}_k - \underline{\mu}_k||_2^2$.

1.1.4. *Empircal Loss Minimization.* By summing our loss function over all the x's, we get a function

$$J(\{\underline{r}_n\}_{n=1}^N, \{\underline{\mu}_k\}_{k=1}^K) = \sum_{n=1}^N \sum_{k=1}^K r_{n,k} ||\underline{x}_n - \underline{\mu}_k||_2^2.$$

Minimizing this is in general difficult because it is nonconvex, and finding a global minimum is NP-hard.

Definition 1.1.5. *The problem of minimizing a given empirical loss function is called the K-Means Problem.*

1.2. Lloyd's algorithm.

(1) First minimize J in terms of each $\underline{r}_n$, only K options. Define $z_n = \text{argmin}_k ||\underline{x}_n - \underline{\mu}_k||_2^2$. Set $\underline{r}_k = e_{z_k}^T$
(2) Minimize J in terms of $\underline{\mu}_k$.

$$||\underline{x}_n = \underline{\mu}_k||^2 = (\underline{x}_n - \underline{\mu}_k)^T \cdot (\underline{x}_n - \underline{\mu}_k)$$

$$\begin{aligned} \nabla_{\mu_k} J &= \nabla_{\mu_k} \sum_{n=1}^N r_{n,k} (\underline{x}_n - \underline{\mu}_k)^T (\underline{x}_n - \underline{\mu}_k) \\ &= -2 \cdot \sum_{n=1}^N r_{n,k} (\underline{x}_n - \underline{\mu}_k) \\ &= -2 \cdot \sum_{n=1}^N r_{n,k} \underline{x}_n - N_k \underline{\mu}_k \end{aligned}$$

with $N_k = \sum_{n=1}^N r_{n,k}, \underline{\mu}_k = \frac{1}{N_k} \sum_{n=1}^N r_{n,k} \underline{x}_n$

Remark 1.2.1. *Setting the gradient to zero tells us*

$$\vec{\mu}_k = \frac{\sum_{n=1}^N r_{n,k} \vec{x}_n}{\sum_{n=1}^N r_{n,k}}$$

To improve this, can use many random initializations. However, we should always use k-Means++, instead of k-Means, which picks one of the data at random, then make a probability distribution, weighted to data that are farthest away from any cluster thus far. Typically, we should standardize the data along each dimension, so that one dimension doesn't everwhelm the others.

Definition 1.2.2. *K-meniods is basically the same as K-means, but the means have to be data points, which make sense in the context where distances arent well defined.*

2. Class 2/3/13

2.1. Section times.

(1) Thursday 1600, MD 223
(2) Friday 1200 MD 123
(3) Friday 1400 NW B150

2.2. Office Hours.

(1) Ryan 2:30 - 4 Mon MD 233
(2) Diana 10-11 Mon MD 334
(3) Sam 3-4 Wed MD 2nd floor lobby
(4) Rest 7-11 Wed Quincy

2.3. Gradient Descent.

Remark 2.3.1. *Idea: to minimize, take steps in the direction of the negative gradient. Suppose we had two parameters, θ_1, θ_2. This might give regions by contour curves . Initialize our algorithm, and try to modify parameters so we reach the global minimum. Let's say our loss function is $L(\theta_1, \theta_2)$. Some methods to minimize L are pertubation, coordinate descent (trying to minimize coordinate one dimension at a time while holding the other dimensions fixed). Kmeans is an example of coordinate descent, but with several blocks at a time. Another type of minimization is gradient descent. The gradient points in most "upward" direction. So, to descent, we would try to go opposite the gradient. Sometimes we can just solve a system to find where the gradient is 0, which is some sort of stationary point. If a convex function has a minimum, it is unique. Much of machine learning is to frame problems to be convex. Simulated annealing is one way to try to solve nonconvex problems.*

Recall kmeans. We have data $\{\vec{x}_n\}_{n=1}^N$, responsibilty vectors $r_{n,k} \in \{0,1\}$ such that $\sum_k r_{n,k} = 1$, mean vectors $\vec{\mu}_k \in \mathbb{R}^D$. There is a loss function $J(\{\vec{\mu}_k\}, \{\{r_{n,k}\}\}) = \sum_{n=1}^N \sum_{k=1}^K r_{n,k} ||\vec{x}_n - \vec{\mu}_k||_2^2$. To get $r_{n,k}$ take those that minimize J by taking the mean closest to each point.

2.4. Kmeans++.

Remark 2.4.1. *The algorithm is as follows. This is just the initialization. After this we can apply Lloyd's algorithm.*

(1) Assign a random $\vec{x}_n$ to $\vec{\mu}_1$.
(2) For i from 1 to K, Draw the mean μ_i distribution of distances to the closest $\vec{\mu}_j$ for $j < k$.

2.5. **Hierarchical Agglomerative Clustering (HAC).** The two main problems with kmeans are

(1) Picking K correctly
(2) Nondeterministic
(3) Data are not disjoint clusters typically

Remark 2.5.1. *HAC is deterministic. Every datum starts in its own group, and you decide how to merge groups. The "action" is the decision of criterion to merge groups. We always merge two objects at once, thus making a binary tree over the data. We can make a dendrogram, which is a visualization in which we draw a sequence of brackets showing merging of data into groups (like a tournament bracket.) The y axis represents points, and x axis represents the distance between two groups before we merge them. Choose K by determining a point after which there is large spacing.*

2.5.2. *Linkage Function.* What this does is determined by the "linkage function." There are four linkage functions from the course notes:

(1) Single: Distance between two groups is the inf over pairs of points. I.e. if $G_1 = \{x_n\}, G_2 = \{y_m\}, D(G_1, G_2) = \min_{m,n} ||x_n - y_m||$
(2) Complete: Minimizing the maximum over pairs of points. $G_1 = \{x_n\}, G_2 = \{y_m\}, D(G_1, G_2) = \max_{m,n} ||x_n - y_m||$
(3) Average: $G_1 = \{x_n\}, G_2 = \{y_m\}, D(G_1, G_2) = \frac{1}{M \cdot N} \sum_{m,n} ||x_n - y_m||$
(4) Centriod: $G_1 = \{x_n\}, G_2 = \{y_m\}, D(G_1, G_2) = ||\frac{1}{N} \sum_n x_n - \frac{1}{M} \sum_m y_m||$

Remark 2.5.3. *If used the single, complete, and average would yield a valid dendrogram, but if we use centroid, might find groups which become more similar, so migth result in "nonvalid" dendrogram. Complete gives compact clusters, single linkage can give "stringy" clusters with long chains.*

3. Class 2/5/14

3.1. **Principal Component Analysis.**

Remark 3.1.1. *Suppose we have some data in $\mathbb{R}^D$ and a limear map $T : \mathbb{R}^D \to \mathbb{R}^K$ with $K < D$. Note that it generally takes $O(n^3)$ time to compute the eigenspecturm.*

There is something called snap-shot method for computing eigenvalues of huge matrices.

One interpretation of PCA is to maximize the variance of the distance from the central point. A second interpretation is to minimize reconstruction error. This is lossy compression, png is lossless, and the image can be reconstructed. png on the other hand loses a lot of information, but the image is quite small, doesn't take much space. PCA is choosing a good lossy compression.

Definition 3.1.2. *An* **orthonormal basis for** $\mathbb{R}^D$ *is a set of D vectors $\{\vec{u}_d\}_{d=1}^D$ such that $\vec{u}_d^T \vec{u}_e = \delta_{de}$.*

Computation 3.1.3. *Assume we have a set of data $\{\vec{x}_n\}_{n=1}^N$ and define $\bar{x} = \frac{1}{N} \sum_{n=1}^N \vec{x}_n$. Then,*

$$x_n = \bar{x} + \sum_{n=1}^N \alpha_d^{(n)} \vec{\mu}_d$$

with $\alpha_d^{(n)} = (\vec{x}_n - \bar{\vec{x}})^T \vec{\mu}_d$*. Then we perform a reconstruction with only* K *basis vectors:*

$$\hat{x}_n = \bar{x} + \sum_{d=1}^{K} \alpha_d^{(n)} \vec{u}_d$$

The squared error is

$$J = \sum_{n=1}^{N} (\vec{x}_n - \hat{\vec{x}}_n)^2$$

3.2. **Computation for PCA.**

Computation 3.2.1. *Try to either preserve variance or minimize reconstruction error. DATA:* $\{\vec{x}_n\}_{n=1}^N, \vec{x}_n \in \mathbb{R}^D$

Game : Find a set of K *orthonormal basis vectors such that we minimize reconstruction error.*

Basis: $\{\vec{\mu}_d\}_{d=1}^K$ *that is orthonormal.*

The mean of the data is $\bar{x} = \frac{1}{N}\sum_{N=1}^{N} \vec{x}_n$.

Write any datum as $\vec{x}_n = \bar{\vec{x}} + \sum_{d=1}^{D} \alpha_d^{(n)} \vec{u}_d$.

Take as our weights $\alpha_d^{(n)} = (\vec{x}_n - \bar{\vec{x}})^T \vec{u}_d$

A compression into $K < D$ *is* $\hat{\vec{x}}_n = \bar{x} + \sum_{d=1}^{K} \alpha_d^{(n)} \vec{u}_d$

Reconstruction loss is $J_n(\{\vec{u}_d\}_{d=1}^K) = (\vec{x}_n - \hat{\vec{x}}_n)^2$.

The Total loss is

$$\begin{aligned}
J_n = (\vec{x}_n - \hat{\vec{x}}_n)^2 &= \left(\left(\bar{\vec{\boldsymbol{x}}} + \sum_{d=1}^{D} \alpha_d^{(n)} \vec{u}_d\right) - \left(\bar{\vec{x}} + \sum_{d=1}^{K} \alpha_d^{(n)} \vec{u}_d\right)\right)^2 \\
&= \left(\sum_{d=k+1}^{D}\right)^2 \\
&= \sum_{d=k+1}^{D} (\alpha_d^{(n)})^2 \\
&= \sum_{d=k+1}^{D} ((\vec{x}_n - \bar{\vec{x}})^T \vec{u}_d)^2 \\
&= \sum_{d=k+1}^{D} \vec{u}_d^T (\vec{x}_n - \bar{\vec{x}})(\vec{x}_n - \bar{\vec{x}})^T \vec{u}_d
\end{aligned}$$

Definition 3.2.2. *The sample covariance is a* D *by* D *positive definite matrix* $\frac{1}{N}\sum_{n=1}^{N} \left(\vec{x}_n - \bar{\vec{x}})(\vec{x}_n - \bar{\vec{x}})^T\right)$

So, summing over all the data, we have

$$\begin{aligned} J &= \sum_{n=1}^{N} \sum_{d=k+1}^{D} \vec{u}_d^T (\vec{x}_n - \bar{\vec{x}})(\vec{x}_n - \bar{\vec{x}})^T \vec{u}_d \\ &= \sum_{n=1}^{N} \sum_{d=k+1}^{D} \vec{u}_d^T \left(\sum_{n=1}^{N} (\vec{x}_n - \bar{\vec{x}})(\vec{x}_n - \bar{\vec{x}})^T \right) \vec{u}_d \\ &= N \sum_{d=k+1}^{D} \vec{u}_d^T \vec{\vec{\Sigma}} \vec{u}_d \end{aligned}$$

Computation 3.2.3. *We want to minimize this subject to $\vec{u}_d^T \vec{u}_d = 1$. Using lagrange multipliers, we want to minimize*

$N \sum_{d=k+1}^{D} \vec{u}_d^T \vec{\vec{\Sigma}} \vec{u}_d + \lambda_d (1 - \vec{u}_d^T \vec{u}_d)$

Taking the gradient, we get $0 = 2\vec{\vec{\Sigma}}\vec{u}_d - 2\lambda_d \vec{u}_d$.

So, $\vec{\vec{\Sigma}}\vec{u}_d = \lambda_d \vec{u}_d$.

Then, we want to minimize $\sum_{d=k+1}^{D} \vec{u}_d^T \vec{\vec{\Sigma}} \vec{u}_d = \sum_{d=k+1}^{D} \vec{u}_d^T \vec{u}_d \lambda_d = \sum_{d=k+1}^{D} \lambda_d$ which corresponds to choosing the largest eigenvectors.

4. Class 2/10/14

4.1. **Unsupervised learning.**

Remark 4.1.1. *In supervised learning we have both inputs and outputs, say $\{x_n, t_n\}_{n=1}^{N}$. The xs are called inputs, features, covariance. The ts are called label, outputs, targets, response, etc. To study this, we will look at*

1. Regression: with $t_n \in \mathbb{R}$. There is also vector regression with $t_n \in \mathbb{R}^m$.
2. Classification: with $t_n \in S$ for S a set.
3. Ordinal Regression: with $t_n \in \mathbb{N}$
4. Structured Prediction: Predicting structured objects, rather than just real numbers

Remark 4.1.2. *Note, the next problem set will probably involve regression. One thing to note about the curse of dimensionality is that in high dimensions, random data tends to all be about the same distance apart. K nearest neighbors predicts elements by their close nieghbors. Three problems with it is noncanonically picking the k, having too many dimensions, and too much data to store.*

5. Class 2/12/14

1. Your gradient is wrong. To correct this use finite differences. Say
$$\nabla_x f(x) = (\frac{d}{dx_i} f(x))_{i=1}^{N}$$
Pick some point x_0. Pick $g(x) = \frac{d}{dx} f(x)$. Note that
$$f(x_0 + \frac{\epsilon}{2}) - f(x_0 - \frac{\epsilon}{2}) \sim \epsilon g(x_0)$$
so you can check this dimensionwise using the gradient. Generally, take $\epsilon \sim 10^{-4}$. Might make a function called CHECKGRAD, and search this on google to find examples.

(2) For trying to solve machine learning problems, try the simplest thing first. Make sure you underfit before overfit.
(3) To debug stuff, generate fake data, and make sure you learn what you're supposed to.
(4) Vectorize and profile your code.

5.1. **Frequentist Regression.**

Remark 5.1.1. *Have inputs $x \in \mathbb{R}^D$ and outputs $t \in \mathbb{R}$, we will have some function $y(x,w) = w_0 + \sum_{i=1}^D w_i x_i$. A basis function regression is when we have $J-1$ basis functions $\phi_j : X \to \mathbb{R}$. We might have basis vectors being polynomials, sinusoids, radial basis functions, etc.*

Definition 5.1.2. *A radial basis funtion is a basis function such that $\phi(x) = \phi(y)$ if $||x|| = ||y||$ In other words, we can write $\phi(x) = \phi(||x||)$*

Computation 5.1.3. *To deal with text documents, we might take a basis function that counts the number of times a words appears. Or, we could ask how similar one document is to some "canonical document."*

$$\text{Let's say we have data } \{x_n, t_n\}_{n=1}^N \text{ and } \phi(x) = \begin{pmatrix} 1 \\ \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_{J-1} \end{pmatrix} \text{ Let } y(x,w) = w^T\phi(x).$$

Then, $\phi : X \to \mathbb{R}^J$ and $w \in \mathbb{R}^J$. Then, define a loss function

$$E_D(w) = \frac{1}{2}\sum_{n=1}^N (t_n - y(x,w))^2 = \frac{1}{2}\sum_{n=1}^N (t_n - w^T\phi(x_n))^2.$$

To minimize this find the gradient $0 = \nabla_w E_D(w) = \sum_{n-1}^N (t_n - w^T\phi(x_n))\phi(x_n) = \sum_{n=1}^N t_n\phi(x_n) - \sum_{n=1}^N \phi(x_n)\phi(x_n)^T w$

Definition 5.1.4. *The design matrix* $\Phi = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_J(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & & \phi_J(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \cdots & \phi_J(x_N) \end{pmatrix}$

Computation 5.1.5. *Then, $0 = \nabla_w E_D(w) = \Phi^T t - \Phi^T\Phi w$ So, we get $\Phi^T t = \Phi^T\Phi w$, so $(\Phi^T\Phi)^{-1}\Phi t = w$ is our condition.*

Definition 5.1.6. $(\Phi^T\Phi)^{-1}\Phi$ *is called the Moore-Penrose pseudoinverse of* Φ

Computation 5.1.7. *Let $t_n = y(x_n w) + \epsilon_n$ for $\epsilon_n \sim N(0, \beta^{-1}$ where β is the precision, which is the inverse variance. Now, $P(t_n|x_n, w, \beta) = N(t_n|y(x_n,w), \beta^{-1})$ $P(t|\Phi, w, \beta) = \prod_{n=1}^N N(t|w^T\phi(x_n), \beta^{-1}) = N(t|\Phi w, \beta^{-1}\mathbb{I}_N)$ with $\mathbb{I}_N$ an $N \times N$ identity matrix.*

6. Class 2/19/14

Remark 6.0.8. *Use star cluster free software from MIT, which is much easier than the raw amazon tools.*

Remark 6.0.9. *Suppose we have Data: $\{x_n, t_n\}_{n=1}^N, t_n \in \mathbb{R}$ and basis functions $\phi_j(x) : X \to \mathbb{R}$. In the practical, much of the challenenge has to do with choosing basis functions. We could, for instance, come up with a list of positive words, and count the number of times these words occur. Suppose we had x's on the left and right and o's in the middle. We could then have functions $\phi_1(x) = x, \phi_2(x) = x^2$.*

Definition 6.0.10. *Suppose we have N data points and J feature functions. Then, the design matrix Φ is an $N \times J$ matrix with $\Phi_{i,j} = \phi_i(x_j)$*

Definition 6.0.11. *Let $y(w, x) = \phi(x)^T w$. Regression is the process of finding a "good" w, which so far has been determined by minimizing a loss function.*

Computation 6.0.12. *The method of least squares says*

$$\begin{aligned} E_D(w) &= \frac{1}{2}\sum_{n=1}^N (t_n - y(x_n, w))^2 \\ &= \frac{1}{2}\sum_{n=1}^N (t_n - \phi(x_n)^T w)^2 \\ &= E_D(w) = \frac{1}{2}(t - \Phi w)^T (t - \Phi w) \end{aligned}$$

Differentiating and solving for w gives $w_{LS} = (\Phi^T\Phi)^{-1}\Phi^T t$

Computation 6.0.13. *Now, let's suppose $t_n = y(w, x_n) + \epsilon_n$ with $\epsilon_n \sim^{i.i.d.} N(0, \beta^{-1})$. Now, let f be the PDF of t_n. Then,*

$$\begin{aligned} f(t_n | w, \phi(x_n), \beta) &= N(t_n | \phi(x_n)^T w, \beta^{-1}) \\ &= \prod_{n=1}^N p(t_n | \phi(x_n)^T w, \beta^{-1}) \end{aligned}$$

Then,

$$\begin{aligned} L(w) &= \log p(t | \Phi, w, \beta) \\ &= \sum_{n=1}^N \log N(t_n | \phi(x_n)^T w, \beta^{-1}) \\ &= \sum_{n=1}^N \left(-\frac{1}{2}\log 2\pi + \frac{1}{2}\log\beta - \frac{\beta}{2}(t_n - \phi(x_n)^T w\right)^2 \\ &= \frac{N}{2}\log 2\pi + \frac{N}{2}\log\beta - \frac{\beta}{2}(t - \Phi w)^T (t - \Phi w) \end{aligned}$$

Then,

$$W_{MLE} = (\Phi^T\Phi)^{-1}\Phi^T t$$

Remark 6.0.14. *Adapting ϕ by applying the chain rule is called* **back propagation.** *This is what is behind neural networks and deep learning, which is really just using the chain rule.*

Computation 6.0.15. *Now, to solve for β, write*

$$L(\beta) = \frac{N}{2}\log\beta - \frac{\beta}{2}(t - \Phi w)^T (t - \Phi w) + C$$

Differentiating, setting to 0, and solving for β,

$$\frac{1}{\beta} = \frac{(t-\Phi w)^T(t-\Phi w)}{N} = \frac{1}{N}\sum_{n=1}^{N}(t_n - \phi(x_n)^T w)^2,$$

with w the w_{MLE}.

7. Class 2/24/14

Remark 7.0.16. *The two estimates given by frequentist methods are point estimates - they predict a single best answer. In Bayesian analysis, we obtain a distribution for our prediction.*

7.1. Bayesian Regression.

Remark 7.1.1. *Let w be a vector of weights, t be outpus, Φ be our matrix of basis functions applied to the inputs, and β be some precision. We'd like to understand $p(w|t, \Phi, \beta)$.*

Theorem 7.1.2. *Let θ be unknown,s and D be data. Then, $p(D|\theta)$ is the likelihood as a function of θ, $p(\theta)$ is ths prior and $p(D)$ is the marginal likelihood. Note that $P(D) = \int P(D|\theta')p(\theta' d\theta'$. Bayes theorem says*

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

Definition 7.1.3. *We say $P(\theta)$ and $P(\theta|D)$ are* **conjugate distributions** *if they are from the same family of distributions. We say $P(\theta)$ is a* **conjugate prior** *to the likelihood function $P(D|\theta)$ if $P(\theta)$ and $P(\theta|D)$ are conjugate distributions.*

Computation 7.1.4. *Now, we can use Bayes Theorem to understand $P(w|t, \Phi, \beta)$*

$$P(w|t, \Phi, \beta) = \frac{P(t|w, \Phi, \beta)P(w)}{P(t|\Phi, \beta)}$$

Let's assume we have likelihood

$$P(t|w, \Phi, \beta) = N(t|\Phi w, \frac{1}{\beta}\mathbb{I}_N)$$

and the prior is of the form

$$p(w) = N(w|m_0, S_0)$$

Then, taking logs, we have

$$\log P(w|t, \Phi, \beta) = \log P(t|\Phi, w, \beta) + \log P(w) - \log P(t|\Phi, \beta).$$

Differentiating with respect to w, and equating to 0, we obtain,

$$0 = \frac{-\beta}{2}(t-\Phi w)^T(t-\Phi w) - \frac{1}{2}(w-m_0)^T S_0^{-1}(w-w_0)$$

After completing the square and exponentiating again, we obtain the posterior distribution satisfies

$$\begin{aligned}\log P(w|t, \Phi, \beta) &= C - \frac{1}{2}\left(\beta t^T t - 2\beta t^T \Phi w + \beta w^T \Phi^T \Phi w + w^T S_0^{-1} w - 2w^T S_0^{-1} m_0 + m_0^T S_0^{-1} m_0\right) \\ &= D - \frac{1}{2}(w-m_N)^T S_N^{-1}(w-m_N)\end{aligned}$$

where C and D are constants in w and

$$m_N = S_N(S_0^{-1}m_0 + \beta\Phi^T t)$$
$$S_N = (S_0^{-1} + \beta\Phi^T\Phi)^{-1}$$

Definition 7.1.5. *The* **Maximimum A Posteriori (MAP) estimation** *is the mode of the posterior distribution.*

Remark 7.1.6. *Using the MAP loses uncertainty, since it represents distributions solely by their peaks.*

Computation 7.1.7. *Say we use the simple prior $P(w) = N(w|0, \alpha^{-1}\mathbb{I})$ Then, let C be a constant in W. We have*

$$\log P(w|t, \Phi, \beta) = C - \frac{\beta}{2}\sum_{n=1}^{N}(t_n - \phi(x_n)^T w)^2 - \frac{\alpha}{2}w^T w.$$

With,

$$S_N = \left(\alpha\mathbb{I} + \beta\Phi^T\Phi\right)^{-1}$$
$$m_N = S_N(\beta\Phi^T t)$$

In this case, our map estimate is m_N since the mode of a gaussian is its mean. The above form is a term for the liklihood, together with a term penalizing large w's, which is a form of regularization.

7.2. Cross Validation.

Definition 7.2.1. *The idea of* **cross validation** *is to split the training data into N parts. Then, for $1 \leq i, leqN$, you choose a part i of the data for validation, and the remaining $N-1$ parts other than part i are used to train for that problem. You then typically take some average of measures of goodness over the N parts to determine the best values of a parameter to use.*

Remark 7.2.2. *Cross validation is very easily parralelizable.*

8. Class 2/26/14

Definition 8.0.3. *A* **hyperparameter** *is a parameter of a prior distribution*

Example 8.1. *In Bayesian linear regression, we can take*

$$P(t|\Phi, w, \beta) = N(t|\Phi w, \frac{1}{\beta}\mathbb{I})$$

$$P(w) = N(w|0, \frac{1}{\alpha}\mathbb{I}).$$

Here, β, w are parameters and α is a hyperparameter.

Definition 8.1.1. *The maximum likelihood value of θ is given by $\theta_{MLE} = argmax_\theta P(D|\theta)$ Then, the* **marginal likelihood**

$$P(D|\alpha) = \int P(D|\theta)P(\theta|\alpha)d\theta.$$

The maximum likelihood estimate for the marginal likelihood is $\alpha_{MLE} = argmax_\alpha P(D|\alpha)$

Remark 8.1.2. *Then, using Bayes theorem we can write*

$$P(\theta|D,\alpha) = \frac{P(D|\theta)P(\theta|\alpha)}{\int P(D|\theta')P(\theta'|\alpha)d\theta'}$$

Note that the denominator here is related to the the marginal liklihood,

Remark 8.1.3. *The MAP estimate is mode of the posterior distribution. We can vaguely model our prior as a uniform distribution with some support Δ_{Prior}. Defining $\theta_{MAP} = argmax_\theta P(D|\theta)P(\theta)$. We then define Δ_{Post} to be the mode of our posterior distribution. We then assume approximately $P(D) = P(D|\theta_{MAP})P(\theta_{MAP})$. We then approximate*

$$P(D) = \int P(D|\theta)P(\theta)d\theta \sim \int P(D|\theta_{MAP})P(\theta)d\theta \sim \int P(D|\theta_{MAP})\frac{1}{\Delta_{Prior}}d\theta \sim P(D|\theta_{MAP})\cdot\frac{\Delta_{Post}}{\Delta_{Prior}}.$$

Where we have $P(\theta) \sim \frac{1}{\Delta_{Prior}}$ because we are assuming the prior is approximately uniform, with probability $P(\theta)$ on its support.

Computation 8.1.4.

$$\begin{aligned} P(t|\Phi,\beta,\alpha) &= \int P(t|\Phi,w,\beta)P(w|\alpha)dw \\ &= \int N(t|\Phi w, \frac{1}{\beta}\mathbb{I})N(w|0,\frac{1}{\alpha}\mathbb{I})dw \\ &= N(t|0, \frac{1}{\alpha}\Phi\Phi^T + \frac{1}{\beta}\Phi) \end{aligned}$$

Remark 8.1.5. *Gaussians are nice, so if $u \sim N(u|\mu,\Sigma)$ Then,*

$$Au = v \sim N(v|Au, A\Sigma A^T)$$

If $\epsilon \sim (0,\Lambda)$ Then,

$$u + \epsilon \sim N(u+\epsilon|\mu, \Sigma + \Lambda)$$

Remark 8.1.6. *If we have some w_{MLE} and β_{MLE} for our most likely coefficients and precision, then $t \sim N(t|\phi(x)^T w_{MLE}, \beta_{MLE}^{-1})$.*

Computation 8.1.7.

$$\begin{aligned} P(t|\phi(x),t,\Phi,\beta,\alpha) &= \int P(t|\phi,w,\beta)P(w|t,\Phi,\beta,\alpha)dw \\ &= \int N(t|\phi^T w, \frac{1}{\beta})N(w|m_N,S_N) \\ &= N(t|\phi^T m_N, \phi^T S_N\phi + \frac{1}{\beta}) \end{aligned}$$

8.2. **Classification.**

Definition 8.2.1. *In machine learning,* **classification** *is assigning to each data element a class in the set $\{C_1,\ldots,C_K\}$.*

Definition 8.2.2. *Data are* **linearly separable** *into 2 classes if there exists an affine space perfectly dividing the data into the correct classifications*

9. Class 3/3/14

Definition 9.0.3. *Take a feature space $X = \mathbb{R}^D$ and labels $t \in \{0, 1\}$. Write $y(x, w, w_0) = w^T x + w_0$. A* **decision boundary** *is a codimension one affineplane defined by $w^T x + w_0 = 0$, and separates our points into those labeled by $t = 1$ and those labeled by $t = 0$.*

Computation 9.0.4. *If x_1, x_2 are in the plane defined by $w^T x + w_0 = 0$, then $w^T x_i = -w_0$ and so*

$$w^T(x_1 - x_2) = w^T x_1 - w^T x_2 = (-w_0) - (-w_0) = 0$$

Intuitively, this tells us that $x_1 - x_2$ is in the plane perpendicular to w.

Computation 9.0.5. *The aim of this computation is to find a vector perpendicular to the plane that intersects it. We know only vectors of the form $c \cdot w$ are perpendicular, where $c \in \mathbb{R}$. Then, in order for the vector to intesect the plane, we need $w^T \left(c \frac{w}{||w||} \right) = c \cdot ||w|| + w_0 = 0$. Solving for c we get $c = \frac{-w_0}{||w||}$. So this is the distnace from the origin to the decision boundary.*

Definition 9.0.6. *A* **one versus all classifier** *is a classifier that tells you whether a certain object is in class k, and it typically used for $k > 2$ classes.*

Remark 9.0.7. *There is ambiguity if we use one versus all classifiers. For instance, if we use decision boundaries, there might be points in the space that wouldn't belong to any of the "correct" sides of our decision boundaries.*

Definition 9.0.8. *A* **one versus one classifier** *for k classes is a set of $\binom{k}{2}$ binary decision classifiers, one for each pair of classes.*

Remark 9.0.9. *Note, this can run into similar problems as the one versus all classifier.*

Definition 9.0.10. *A* **Non-ambiguous Multi-class classifier** *is a set of k functions $y_k(x, w_k, w_{k0}) = w_k^T x + w_{k0}$, and x is in class k if $k = argmax_k y_k(x, w_k, w_{k0})$.*

Remark 9.0.11. *This is ill defined where two of the functions both have the maximum value, but this is typically a set of measure 0.*

Definition 9.0.12. **Fisher's linear discriminant** *is a function of the form $y(x, w, w_0) = w^T x + w_0$, so that w minimizes the function*

$$J(w) = \frac{-(m_1 - m_2)^2}{v_1 + v_2}$$

where m_i is the mean of the projected data $w^T X_i$ and v_i is the variance of the porjected data $w^T X_i$.

Computation 9.0.13. *Suppose we have data distributed according to $X_1 \sim N(x_1 | m_1, S_1)$, $X_2 \sim N(x_2 | m_2, S_2)$. Then, after projecting, we have $w^T X_i \sim N(w^T x_i | w^T m_1, w^T S_1 w)$.*

Computation 9.0.14.

$$J(w) = \frac{-(w^T m_1 - w^T m_2)^2}{w^T S_1 w + w^T S_2 w} = \frac{-w^T (m_1 - m_w)(m_1 - m_w)^T w}{w^T (S_1 + S_2) w}.$$

In order to minimize J, we differentiate, set equal to 0, and then solve for w. Define $S_w = S_1 + S_2, S_b = (m_1 - m_2)(m_1 - m_2)^T$. Then, $J(w) = \frac{-w^T S_b w}{w^T S_w w}$

$$0 = J'(w) = -\frac{(2S_b w)(w^T S_w w) - (2S_w w)(w^T S_b w)}{(w^T S_w w)^2}$$

Solving, since $w^T S_w w, w^T S_b w$ are scalars, we need $S_b w \propto S_w w$. That is, we need $(m_1 - m_2)(m_1 - m_2)^T w \propto S_w w$, so $(m_1 - m_2) \propto S_w w$, since again $(m_1 - m_2)^T w$ is a scalar. Hence, we want

$$w \propto S_w^{-1}(m_1 - m_2).$$

9.1. **Perceptrons.**

Remark 9.1.1. *A general theme in machine learning is as followsGiven some features $\{x_1, \ldots, x_d\}$, we compute a weighted sum $S_w(x) = \sum_{d=1} w_i x_i$, apply some nonlinear function $f(S_w(x))$ and obtain some result.*

Definition 9.1.2. *Define the function* $f(a) = \begin{cases} 1 \text{ if } a \geq 0 \\ -1 \text{ if } a < 0 \end{cases}$ *A* **perceptron** *is the function $f(w^T x_n)$.*

Definition 9.1.3. *A* **perceptron error function,** *for the perceptron defined above is $E_P(w) = -\sum_{n=1}^N f(w^T x_n) t_n$.*

Remark 9.1.4. *Suppose we have data $\{x_n, t_n\}_{n=1}^N$ and error function $E_w(x) = -\sum_{n=1}^N t_n f(w^T x_n)$. Typically we would perform gradient descent on this error function. That doesn't work here because f is not differentiable.*

Definition 9.1.5. *The* **Perceptron Learning Algorithm** *is an algorithm for classifying data using perceptrons. Explicitly, For every piece of data, write $a = w^T x_n + w_0$ and if $at_n \leq 0$, then set $w = w + t_n x_n, w_0 = w_0 + 1$.*

Remark 9.1.6. *It's easy to show this converges if the data is linearly separable. Obviously if it is not linearly separable, it will never converge.*

Remark 9.1.7. *Perceptrons can't recognize the exor function, because it is not linearly separable. To rectify this, we introduce multi-layer perceptrons, or neural networks.*

Definition 9.1.8. *A* **class conditional.** *which is the conditional probability distribution of x given c is in class k, which we shall notate $P(x|c = k)$.*

Computation 9.1.9. *Say we have a prior probability that an element c is in class k, which we shall notate $P(c = k)$. Suppose we also have class conditionals $P(x|c = k)$. Then, using Bayes theorem,*

$$P(c = k|x) = \frac{P(c = k)P(x|c = k)}{\sum_{j=1}^K P(c = j)P(x|c = j)}$$

Definition 9.1.10. *The method of* **Generative Classification** *is given by the process in the previous computation, by which we start with priors and class conditionals, and use them to predict the probability an element is in a certain class.*

Definition 9.1.11. *A* **Naive Bayes classifier** *is a classifier for x assuming all of its components x_d are independent. That is, we take*

$$P(x|c=k) = \prod_{d=1}^{D} P(x_d|c=k),$$

and then use generative classifiers for predicting each of the $P(x_d|c=k)$ separately.

Remark 9.1.12. *Generative models try to compute the distribution for the x's. It might waste effort if we just want to be able to classify them.*

Definition 9.1.13. *A* **Sigmoid function** *is heuristically an S shaped function.*

Definition 9.1.14. *The* **logistic sigmoid function** *is $\sigma(x) = \frac{1}{1+e^{-x}}$.*

Computation 9.1.15. *Define $a = \log \frac{P(c=1)P(x|c=1)}{P(c=0)P(x|c=0)}$. Then,*

$$\begin{aligned} P(c=1|x) &= \frac{P(c=1)P(x|c=1)}{P(c=1)P(x|c=1)+P(c=0)P(x|c=0)} \\ &= \frac{\frac{P(c=1)P(x|c=1)}{P(c=0)P(x|c=0)}}{\frac{P(c=1)P(x|c=1)+P(c=0)P(x|c=0)}{P(c=0)P(x|c=0)}} \\ &= \frac{\frac{P(c=1)P(x|c=1)}{P(c=0)P(x|c=0)}}{1+\frac{P(c=1)P(x|c=1)}{P(c=0)P(x|c=0)}} \\ &= \frac{e^a}{1+e^a} \\ &= \frac{1}{1+e^{-a}} \end{aligned}$$

10. Class 3/10/14

10.1. Logistic Regression.

Remark 10.1.1. *Let be the probability that t is class 1. Then, as we saw last time,*

$$P(t=1|x,\Sigma,\mu_1,\mu_2,\rho) = \frac{1}{1+e^{-a}}$$

where $a = w^t w + w_0, w = \Sigma^{-1}(\mu_1-\mu_2), w_0 = \frac{-1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \log\frac{\rho}{1-\rho}$ In the case $\mu_1 = \mu_2$, we see that the decision is completely determined by the prior, as we expect.

Remark 10.1.2. *The generative model computes $O(D^2)$ entries of the covariance matrix, which may be expensive. The benefit we gain from this is that we can generate data. Logistic regression just tries to optimize w, w_0, but it loses the ability to generate data. The aim is to find the w that optimize the labels given the parameters w.*

Computation 10.1.3. *Say we have data $\{x_n, t_n\}, t_n \in \{0,1\}$ Then,*

$$\begin{aligned} P(\{t_n\}|\{x_n\}, w) &= \prod_{n=1}^{N} P(t_n|x_n, w) \\ &= \prod_{n=1}^{N} \sigma(x_n^T w)_n^t (1-\sigma(x_n^T w))^{1-t_n} v \end{aligned}$$

Then,

$$\log P(t_n|\{x_n\}, w) = \sum_{n=1}^{N} t_n \log \sigma(x_n w) + (1-t_n)\log(1-\sigma(x_n^T w)).$$

Using $1-\sigma(z) = \sigma(-z)$, we can see

$$\frac{d}{dz}\sigma(z) = \sigma(z)\cdot(1-\sigma(z))$$

Differentiating to use gradient ascent, we have

$$\begin{aligned} \nabla_w \log P(D|w) &= \sum_{n=1}^{N} t_n \frac{1}{\sigma(x_n^T w)}\sigma(x_n^T w)(1-\sigma(x_n^T w) - (1-t_n)\frac{1}{1-\sigma(x_n^T w)}\sigma(x_n^T w)(1-\sigma(x_n^T w))x_n \\ &= \sum_{n=1}^{N}(t_n - \sigma(x_n^T w))x_n \end{aligned}$$

We then update using $w_{new} = w_{old} + \alpha \nabla_w \log P(\{t_n\}|\{x_n\}, w)$.

10.2. **Neural Networks.**

Computation 10.2.1. *The idea of Neural Networks is that instead of just taking normal basis functions, we allow the basis functions to depend on additional parameters θ_j. So suppose we have basis functions with $\phi_j(x, \theta_j)$, and predicting function $y(x, w, \theta) = \sum_{j=1}^{J} \phi_j(x) w_j$. Then, define $E_n(w) = \frac{1}{2}\left(t_n - \sum_{j=1}^{J}\phi_j(x)w_j\right)^2$, and $a_n = \sum_{j=1}^{J}\phi_j(x_n)w_j$ Observe that*

$$\frac{\partial}{\partial w_j}E_n = \frac{\partial E_n}{\partial a_n}\frac{\partial a_n}{\partial w_j}.$$

To do gradient descient, we have

$$w_j^{new} = w_j^{old} - \alpha \frac{\partial}{\partial w_j}\sum_{n=1}^{N} E_n.$$

Similarly, we can use gradient descent for the θ_j parameters:

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j}\sum_{n=1}^{N} E_n.$$

Note then that

$$\frac{\partial}{\partial \theta_j}E_n = \frac{\partial E_n}{\partial a_n}\frac{\partial a_n}{\partial \phi_{jn}}\frac{\partial \phi_{jn}}{\partial \theta_j}.$$

Observe that $\frac{\partial a_n}{\partial \phi_{jn}} = w_j$

Definition 10.2.2. *A* **hidden layer** *is a set of Given some inputs $x_1, \ldots, x_D$, and weight vectors $h_1, (w^1, x), \ldots, h_J(w^J, x)$, together with a function $\sum_{j=1}^J v^j \sigma(h_j(w^j, x)$ where σ is some sigmoid function, and and the w^j, v^j are fixed vectors, and the h_j are various basis functions.*

Definition 10.2.3. *A* **Neural Network** *is a collection of hidden layers, togeter with maps taking the output of one hidden layer as the inputs for the next hidden layer.*

11. Class 3/12/14

11.1. **Decision Trees.**

Definition 11.1.1. *First, let me give a formal definition: A* **decision tree** *is a classification function f represented by a collection of decision functions $g_i, i \in I$ and end nodes $h_j, j \in J$, where each h_j is a classification. Then $f(x)$ is defined as follows: define $i_1 = g_1(x)$, and recursively define $i_n = g_{i_{n-1}}(x)$, so long as $i_n \in I$. If $i_n \notin I$, then we require $i_n \in J$. Let $j = i_n$, where n is the least integer for which $i_n \notin I$. Then define $f(x) = h_{i_n}$.*

Remark 11.1.2. *The definition above is highly formal and fairly opaque. The idea is that we have a tree, where we put in our input at the top, and follow the arrows from one level to the next which describes the input x. Satisfying the following properties:*

1. *Every internal note has one attribute*
2. *Branches occur on different attribute values*
3. *Each attribute appears at most once in a path*
4. *At a leaf node, return a label*

Remark 11.1.3. *In theory we could enumerate every possible decision tree and pick the ones closest to our data. However, if we have d attributes, there are 2^d possible classifications for each elements of our data based on these attributes, and hence at least 2^{2^d} total possible labelings for for all the data (since each data can be classified in at least two different values.) This is far too big, so we'll need to use information theory to determine which attributes are important.*

Definition 11.1.4. *Let X be a discrete random variables. The* **information content** *denoted*

$$I(X = j) = \log_2 \frac{1}{P(X = j)}.$$

Definition 11.1.5. *The* **Shannon Entropy** *of a discrete random variable X is*

$$H(X) = E[I(X = j)] = -\sum_{j=1}^{J} P(X = j) \log \frac{1}{P(X = j)}.$$

Example 11.2. *In the case $X \sim Bern(p)$ we obatin $H(X) = -(p \log p + (1 - p) \log(1 - p))$. Observe that this is maximized at $p = .5$, with $H(X) = 1$, but as p approaches 0 or 1, the entropy approaches 0.*

Definition 11.2.1. *Let X, Y be two random variables. The* **specific conditional entropy** *of X given $Y = k$ is $H(X|Y = k) = -\sum_{j=1}^J P(X = j|Y = k) \log P(X = j|Y = k)$,*

Definition 11.2.2. *For X, Y two random variable, the* **conditional entropy** *of X given Y is*

$$H(X|Y) = \sum_{k=1}^{K} P(Y=k)H(X|Y=k) = -\sum_{k=1}^{K} p(Y=k)\sum_{j=1}^{J} P(X=j|Y=k)\log P(X=j|Y=k).$$

Definition 11.2.3. *The* **mutual information** *of X and Y is*

$$I(X;Y) = \sum_{j=1}^{J}\sum_{j=1}^{K} P(X=j, Y=k)\log\frac{P(X=j, Y=k)}{P(X=j)P(Y=k)}.$$

Exercise 11.3. *Show $I(X;Y) = I(Y;X)$ and $I(X;Y) = H(X)+H(Y)-H(X,Y), I(X;Y) = H(X) - H(X|Y)$*

Remark 11.3.1. *Once you are able to calculate entropies, you want to pick attributes to minimize entropy, and you might stop after some given depth.*