AI II NOTES

BRENDAN WHITAKER

2010 *Mathematics Subject Classification.* 12-XX

Abstract. Notes for Professor Jihun Hamm's AI II course, taken SP18.

Contents

Part 1

CHAPTER 1

LEMMA 1.1. $E[x] = E[E[X|Y]]$.

PROOF. Observe:

$$\begin{aligned} E[E[X|Y]] &= \sum_i E[X|y = y_i]P(Y = y_i) \\ &= \sum_i \sum_j x_j P(X = x_j|Y = y_i)P(Y = y_i) \\ &= \sum_i \sum_j x_j P(X = x_j, Y = y_i) \\ &= \sum_j x_j \sum_i P(X = x_j|Y = y_i) \\ &= \sum_j x_j P(X = x_j) \\ &= E[X]. \end{aligned} \tag{1.1}$$

□

LEMMA 1.2. $Cov[X, Y] = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y]$.

PROOF. Observe:

$$\begin{aligned} E[(X - E[X])(Y - E[Y])] &= E[XY - XE[Y] - YE[X] + E[X]E[Y]] \\ &= E[XY] - E[X]E[Y] - E[Y]E[X] + E[X]E[Y] \\ &= E[XY] - E[X]E[Y]. \end{aligned} \tag{1.2}$$

□

LEMMA 1.3. *If two random variables are independent, then they are uncorrelated.*

PROOF. Recall that X, Y are independent if and only if $P(X, Y) = P(X)P(Y)$. And recall that X, Y are **uncorrelated** if $E(XY) = E(X)E(Y)$. Observe:

$$\begin{aligned} E(XY) &= \sum_x \sum_y xyP(X = x, Y = y) \\ &= \sum_x \sum_y xyP(X = x)P(Y = y) \\ &= \sum_x xP(X = x) \sum_y yP(Y = y) \\ &= E(X)E(Y). \end{aligned} \tag{1.3}$$

□

REMARK 1.4. For a Gaussian random variable, **uncorrelated** implies independent.

DEFINITION 1.5. We can combine multiple densities (pdf) to create a new density using a **convex combination** of $P_i(x)$:

$$P(x) = a_1P_1(x) + \cdots + a_nP_n(x),$$

where we must have $a_i \geq 0$ and $a_1 + \cdots a_n = 1$.

Remark 1.6. This is **NOT** the same as a convex combination of random variables.

Definition 1.7. A **mixture of Gaussians (MOG)** is this sum of densities with Gaussian distributions.

Definition 1.8. For mixture of gaussians, what is the probability that class j generated the given point? It is:

$$\begin{aligned} P_{ij} = P(C=j|X=x_i) &= \frac{P(X=x_i|C=j)}{P(X=x_i)}P(C=j) \\ &= \alpha P(X=x|i|C=j) \\ &= \textit{something...} \end{aligned} \tag{1.4}$$

Definition 1.9. We give a formula for **KL divergence**:

$$KL(q||p) = E_q\left(\frac{q(Z)}{p(Z)}\right).$$

where E_q is the expected value over q and $q(Z)$ is the density of q.

Example 1.10. We compute $KL(N(0,1)||N(1,1))$. Let Q be $N(0,1)$ and let $P = N(1,1)$. Observe:

$$\begin{aligned} KL(N(0,1)||N(1,1)) &= \int Q(X)\log\frac{Q(X)}{P(X)}dX \\ &= \int \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}\cdots \end{aligned} \tag{1.5}$$

CHAPTER 2

Exam review

Lecture 1 is just syllabus, so no questions for that. We are up to decoding in hmm. We should have 6 questions. It's closed book, closed note, no calculators necessary. Some will be multiple choice, some will be short answer. He might put in missing blanks, but he won't ask us long derivations. The midterm will definitely be easier than homework. We will have solutions for homework.

2.1. lecture 2

Definition 2.1. Memorize the univariate Gaussian density.

Remark 2.2. If he asks us questions about multivariate bivariate, he will give us distribution.

Remark 2.3. Marginalization and product rule for bayesian networks.

Remark 2.4. basic probability theorems and computations, independence, bayes rule, product rule.

Definition 2.5. Naive bayes model, if mutually independent given the common cause, then we have a product decomposition.

Forget about spam filter example.

2.2. Lecture 3

Know the burglary bayesian network. Know how to compute probabilities using structure and conditional probabilities. Also for the car diagnosis example as well. If he supplies CPT for car diagnosis, you should be able to compute probabilities. Will only get as big as 5 nodes ish. Arrow doesn't necessarily mean temporal cause.

Remark 2.6. Product of nodes conditioned on parents is the whole joint probability, this is the essence of bayesian network.

Remark 2.7. Be familiar with examples from homework 1.

Remark 2.8. Skip node ordering.

Remark 2.9. Know d-separation.

Condition 3 is the only thing you can use when there is nothing observed.
Know definition of markov blanket.

Remark 2.10. Skip continuous nodes and linear gaussian models in context of bayesian networks.

Definition 2.11. D separation doesnt hold for markov blankets. Its only for bayesian networks. As long as theres an observed node between two nodes, they are blocked, it's independent.

Remark 2.12. Skip de-noising example.

If he asks us question on markov network it would be simpler.
Skip everything after that, conversion between markov and bayesian, the rest.

2.3. lecture 5

Remark 2.13. If MLE, you will be able to take deriv and log and solve normally.

Remark 2.14. Know bayesian estimation, for bernoulli, and gaussian where mean is also gaussian.

Remark 2.15. Make sure you can do the derivation on slide 24.

Complete the square. Email him about figuring out this derivation.
Skip text modelling and spam mail and SKip everything past slide 25.

2.4. lecture 6

Know expected value and conditional expectation definitions.

Know E[x] and $E[x^2]$ of gaussian derivations.

Know expected values are linear.

Know law of total expectation.

$E[X|Y]$ can be RV, random with Y. But $E[X]$ is just a constant in X. So note $E[E[X|Y]$, the outer expectation is with respect to the randomness of Y, not X, so you multiply by $P(Y)$.

DEFINITION 2.16. Know independence vs uncorrelatedness.

DEFINITION 2.17. Know basic mixture model equation, and compute log likelihood.

REMARK 2.18. EM algorithm will be on exam. There's no avoiding studying it, look at it for mixture of gaussians. We're estimating the membership function. Mixture model has three parameters, mean, covariance, and class prior. P_{ic} is the membership function computed in E step.

Is the covariance matrix of rank 1? Is that waht he said? He won't ask us to derive M-step.

Question, what would he ask us to do if not the derivation of M step/E step.

REMARK 2.19. If you take what we did for M step in homework and maximize like in MLE, you will complete M-step.

REMARK 2.20. KL divergence. Know it.

REMARK 2.21. Skip jensen's inequality.

Skip EM increases log likelihood slide, proof of EM, skip it all. Try to understand and memorize the general form/formula though.

Theta old is random for first iteration. You take global mean and shit, refer to MOG initialization slide.

Ask for an EM algorithm example problem via email.

K means clustering will be on exam. Everything on homework will be possible. But lengthy derivations maybe not. Not line homework 1 question 1.

bayesian estimation for coin toss is too long. For m + 1, too long.

Memorize mixture model formula. Don't need to know other distribution definitions. Memorize KL divergence.

2.5. Lecture 7

Skip langauge modelling.

Think about other examples of HMM like dishonest casino. Understand HMM.

Not going to ask decoding eval and learning probelems directly.

Know definition of markov model.

Should be able to sample points with bayesian network using HMM.

So know how to do 16-18 for a question.

Rest of HMM will be covered later other exam.

CHAPTER 3

Machine Learning

3.1. Intro to Machine Learning

Friday, March 9th

Your data consists of the features or attributes and the label. Say we're trying to classify cat or dog from images. So the image is a real valued matrix. The dataset is associated with some distribution. So we have a distribution:

$$P(X|Y = D), P(X|Y = C), P(Y = D).$$

Where $X = \mathbb{R}^{1000}$ and $Y = \{ C, D \}$. We assume training set and testing set are i.i.d. samples from the same distribution. In kaggle, they give you a training set, and you're free to use your own models. The administrator of the test will take your models and run it on their own test set that's hidden from you. The test set is something that you never see. As long as you don't use test set to get feedback, it's fine. Ideally, it should never be involved in the training procedure. Training set alone is not enough to build a good model. You're not sure which model works best for the given data. You have multiple ones that might work. Say we try a 2nd order polynomial and a 5th order and a 10th order. We use common training data to fit the best polynomial for all 3. Then we have 3 models trained on the same training data. Then we evaluate the 3 models on the evaluation set, and compare the results. Among the three, the one with smallest error is a good candidate for submission. So we have 3 types of sets, the one we use to pick from the three models is called the public test set or evaluation set, the private test set is not known. A popular approach is called n-fold cross validation. For 2-fold cross validation, you train with one half and test with the other, then you do the opposite. Then you simply average the accuracies and use that as a measure.

Logistic regression has a regulation coefficient. You have to find out the ideal values for these parameters by using k-fold cross validation. If you use test set to revise accuracy/model, then your reported accuracy cannot be trusted since you used some information from your testing set to revise your model. That's how it should be, ideally. But in reality, if your algorithm works very well on the validation set, it's going to work very well on the test set as well. But let's try to be pedagogical and stick with best practices. If you have any experience with machine learning, you should have heard these terms before.

Slides:

Choose a performance measure and hypothesis class. Your choice of these two objects determines what model you're going to use.

Today we're going to discuss some classic models of supervised learning. Let's talk about linear regression. So we're trying to predict house price from the size of the house. We're on slide 25. Of course we can't perfectly model all these points. We could use an n-th degree polynomial to pass through each point, but it's ill-advised. Slide 26. We're using a set of linear lines as our hypothesis class H. The parameter of the linear line is w_0, w_1 or denoted $w = \langle w_0, w_1 \rangle$. The training error is given by the loss function. The loss is $L_2 = |y_{pred} - y_{true}|^2$. The training procedure is to find the best parameter set that minimizes the training error. But can we find these optimal lines analytically? Of course, it's definitely a doable problem.

The training error is:

$$\begin{aligned} Err &= \frac{1}{N} \sum_j L(y_j, f(x_j)) \\ &= \frac{1}{N} \sum_j (y_i - w_1 x_j - w_0)^2. \end{aligned} \tag{3.1}$$

So we have:

$$\begin{aligned}\frac{\partial Err}{\partial w_0} &= \frac{-2}{N}\sum_j (y_j - w_1 x_j - w_0) = 0 \\ &= \sum_j (\text{""}) = 0 \\ &= \sum_j y_j - w_1 \sum_j x_j - N w_0 = 0.\end{aligned} \tag{3.2}$$

Then we'd solve the other partial equation, and then solve for w_1, w_0.

So for the n-dimensional case, we introduce vector notations. The attribute is usually a vector. Sometimes we will use boldface to distinguish a vector from a nonvector, Recall the dot product $\mathbf{w}\cdot\mathbf{v}$. Using this notation, the training error is:

$$Err(\mathbf{w}) = \frac{1}{N}\sum_j (y_j - \mathbf{w}\cdot\mathbf{x_j})^2. \tag{3.3}$$

We will also write vectors as vertical, single-column matrices.

We have training labels Y, and $N \times 1$ matrix. And our observations are X, and $N \times D$ matrix where D is the dimension of the problem. We have:

$$X = \begin{bmatrix} x_1^T \\ \vdots \\ x_N^T \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \tag{3.4}$$

Where the y_i's are scalars, but the x_i's are $D \times 1$ vectors, so their transposes are $1 \times D$.

We have:

$$\begin{aligned} Err(\mathbf{w}) &= (Y - X\mathbf{w})^T(Y - X\mathbf{w}) = (Y^T - \mathbf{w}^T X^T)(Y - X\mathbf{w}) \\ &= Y^T Y - Y^T X\mathbf{w} - \mathbf{w}^T X^T Y + \mathbf{w}^T X^T X\mathbf{w}.\end{aligned} \tag{3.5}$$

Then we have:

$$\begin{aligned}\nabla_{\mathbf{w}} Err &= 0 - X^T Y - XY + 2X^T X\mathbf{w} \\ &= 2X^T Y + 2X^T X\mathbf{w} = 0.\end{aligned} \tag{3.6}$$

Now we discuss gradient descent. We pick a point, and then compute slope at this point, then shift the point downwards by a constant proportional to the absolute value of the slope, and eventually you will find local min.

So for D-dimensional case, we compute the gradient vector, which is the slope of the steepest ascent, and then you do the same thing, you will reach the bottom of the D-dimensional surface and your step size will drop until it hits zero. The equation is:

$$\mathbf{x} \longleftarrow \mathbf{x} - \eta\nabla\mathbf{f}.$$

Note we're moving in the negative gradient, since the gradient is defined as the direction of steepest ascent. Now we want to apply the gradient descent to minimizing the training error. If $\mathbf{w}$ is the parameter of our model, we use the update rule:

$$\mathbf{w} \longleftarrow \mathbf{w} - \alpha\nabla_{\mathbf{w}} Err, \tag{3.7}$$

where $\alpha > 0$ is the learning rate. There will probably be a homework problem for this. So this update rule can just be written as an elementwise scalar update rule for arbitrary dimensionality. The algorithm is just repeating this rule. Recall that:

$$\nabla_{\mathbf{w}} Err = \frac{1}{N}\left[\sum_j \nabla_{\mathbf{w}} L(f(\mathbf{x}_j), y_j)\right].$$

So we have to choose an α, and we don't know which is better, but the model is highly sensitive to which one we choose. So we need to use an evaluation set to decide which to choose.

For quadratic problems like linear regression, there is an ideal step size. In general, if it's a convex function, there is a good choice of α, but for general functions, you can't analytically find it. If it's convex, you could mis-specify this value, it will still converge, just slow as heck. For non-convex functions, a bad choice of your learning rate could throw you way far away from your true solution and you'd have no idea. Will this always converge in a finite number of steps? Probably not, we stop after a preset number of iterations or when the error is smaller. If slope is small enough, you know it's not gonna make much progress anymore, so you stop.

So when you're updating your $\mathbf{w}$ using the gradient of this Error function, you compute the gradient of the log for each sample, and you take the sample average of the gradient, and you follow that direction. All we did was change the order of the gradient and the summation, which we can always do.

So what we had was using gradients averaged over samples:

$$\mathbf{w} \longleftarrow \mathbf{w} - \alpha \frac{1}{N}\left[\sum_j \nabla_{\mathbf{w}} L(f(\mathbf{x}_j), y_j)\right]. \tag{3.8}$$

This is inefficient for large samples. So define **stochastic gradient descent**:

$$\mathbf{w} \longleftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L(f(\mathbf{x}_j), y_j). \tag{3.9}$$

It uses one sample at a time. Recall:

$$\begin{aligned}\nabla_{\mathbf{w}} L(y_j, f(x_j)) &= ||y_j - \mathbf{w}^T x_j||^2 \\ &= \nabla_{\mathbf{w}}(...)(...) \\ &= ...\end{aligned} \tag{3.10}$$

But we get the same result as on slide 29, or at least something analogous. We have:

$$\nabla_{\mathbf{w}} L = -2\mathbf{x}_j y_j - 2\mathbf{x}^T x_j w. \tag{3.11}$$

If this is in any way confusing, it's probably because it's a vector gradient. It's really nothing. Same as basic calculus. If you can, compute the gradient of any loss function, and use that. If you use a different learning algorithm as long as you can compute the gradient. And we will do that in the coming weeks. Now we'll check out a demo.

So back to linear regression in the one-dimensional case. We're going to use gradient descent. The error function is a paraboloid. The first step is the largest because the gradient is steepest, and then after that they get much smaller, and eventually the points become arbitrarily close to the bottom of the surface. For other problems, the surface may not be convex, and thus it may not be guaranteed to converge.

Note that the Wednesday after we come back from break, we need a project proposal.

Wednesday, March 21st

We review linear regression. Our loss function is usually $L_2 = |y_{pred} - y_{true}|^2$. We define training error as:

$$Err(w) = \frac{1}{N}\sum_j (y_j - f(x_j))^2.$$

We want to find a unique minimum for our Err(w) which we will do by computing the gradient of the error function. We will find the solution w_0, w_1 which corresponds to the min error. We set the component-wise derivatives equal to zero and solve. In general, the training error is given by:

$$\begin{aligned}Err(\mathbf{w}) &= \frac{1}{N}(y_J - f(x_j))^2 = \frac{1}{N}\sum_j (y_j - \mathbf{w}\cdot\mathbf{x_j})^2 \\ &= \frac{1}{N}||y - X\mathbf{w}||^2 \\ &=\end{aligned} \tag{3.12}$$

Note this method doesn't work for high-dimensionality, since its running time is proportional to n^3. A better way for big data is to use **gradient descent**. Recall the max value of the negative of the gradient descent is the location where the surface is steepest. So gradient descent is applied to minimize training error. All we need to do is use the update rule:

$$\mathbf{w} \longleftarrow \mathbf{w} - \alpha \nabla_w Err.$$

where $\alpha > 0$ is the learning rate. Note the gradient of the whole error function can be written as the average of the gradient of each instance of the loss function L, by linearity of the derivative.

So to deal with a series of 3 parabolas, we can first average the parabolas to one big parabola and take the gradient there, or we can take the gradient of each parabola and then average the gradients. This is linearity of the gradient in action. So there is a variation of gradient descent called **stochastic gradient descent.** This method uses one sample at a time instead of gradients averaged over samples, since often the averages are computationally expensive:

$$\mathbf{w} \longleftarrow \mathbf{w} - \alpha \nabla_w L(f(\mathbf{x}_j), y_j).$$

This is not the actual gradient direction, it is a noisy direction. They are only the same in the expectation. If we use full gradient descent, we take time to compute gradient of all examples and will point us to the steepest direction. And $\mathbf{w}$ is updated just once. Now with stochastic gradient descent, we don't wait until we have computed the whole gradient, it's like implementing a smaller step size: we update $\mathbf{w}$ much more often. Computing N iterations of stochastic gradient descent takes just as long as a single iteration of gradient descent. There is a middle ground called "mini-batch" gradient descent which does k samples at a time. This middle ground is actually the best, more efficient than either of the other purist approaches.

3.2. Linear classification

Now the feature is two-dimensional. We wish to separate two or more classes using a hyperplane. Is it linearly separable? If not we call it **linearly non-separable**. The separating line is given by:

$$h(x) = \mathbf{w} \cdot \mathbf{x} = w_0 + w_1 x_1 + w_2 x_2 = 0.$$

Note a hyperplane is a $d-1$-dimensional subspace of $\mathbb{R}^d$. In $\mathbb{R}^3$, the hyperplane is any line. We define our loss function: $L = 0$ if $y = f(\mathbf{x})$ and 1 if $y \neq f(\mathbf{x})$. We define a threshold function $g : \mathbb{R} \to \mathbb{R}$ which looks like a square sigmoid on $[0, 1]$. We decide to use ± 1 for the label instead of $0, 1$. We multiply the true label $-y$ by $\mathbf{w} \cdot \mathbf{x}$ to get an alternative definition of the loss function, but it's equivalent. The third step of training is to minimize the training error. Our error function looks very similar:

$$Err(w) = \frac{1}{N} \sum_j L_{0/1}(y_j, f(\mathbf{x_j})) = \frac{1}{N} \sum_j g(-y \cdot (\mathbf{w} \cdot \mathbf{x_j})).$$

In general it is not easy to find the minimum of this error function since it may not be a convex function. Linear classification is one of the simplest classification methods we can think of, but it's actually quite difficult to solve. Note that the threshold is non-differentiable, since it's a piecewise/square function. So we use a sigmoid! The function is given by:

$$g(t) = \frac{1}{1 + e^{-t}}.$$

Note as $t \to \infty$ the function goes to one, and the function goes to zero as $t \to -\infty$, and as $t \to 0$, the function approaches $\frac{1}{2}$. Also known as the logistic function. The loss function uses logistic loss. It is a homework problem to compute the gradient of the loss function. The answer is given in the slides, but note this is not a univariate problem, we need to use multivariate chain rule. This chain rule is given. Basically, when you have composition of functions that are of several variables, you need to take the gradient of them in the chain rule. If you want a challenging problem, the loss function is actually convex. It has a unique minimum value. Maybe we will try and prove it later. This is a much nicer model than the linear classifier, and the overall approach is known as **logistic regression**. A better name for this would be called logistic classifier.

3.3. Support vector machines

More than one hyperplane can exists that perfectly separates positive and negative samples. We will not go into too much depth about the inner workings of SVM. Think of this as a preview of the in-depth explanation of SVM. Imagine you have two sets of points, positive and negative, and you want to separate them using a linear hyperplane. And there are often infinitely many such hyperplanes. The training error is zero for all such lines. So which is best?

In fact there is a geometric solution to find the best. Try moving a line along it's normal directions until it touches any point.

Definition 3.1. We define the margin as half the distance between the lines which are tangent to the nearest points when moving out from the original line in the normal directions.

We claim that a larger margin is better than a smaller margin. Intuitively, if the smaller margin is tested on other data, we have a greater chance of misclassification since we have a low margin between where we think the two classes are centered and where the decision boundary is.

So our goal is to find the hyperplane with the maximum margin. Alright, so let's derive the formula for the margin. The equation $f(x) = 0$ is our hyperplane. The region above the line is where our function f is positive, and it's negative below it. Consider a vector $\mathbf{w}$ normal to our hyperplane f.

Definition 3.2. We define our signed distance from the hyperplane as $f(x)/||\mathbf{w}||$. We define the training point closest to the hyperplane, also known as the **support vector** as the point which gives $\min_p y_p f(x_p)/||\mathbf{w}||$. Recall y_p is just the label of the points in the training set, so $y_p \in \{ \pm 1 \}$. The value of this minimum is the margin.

So the overall problem is to find $\mathbf{w}$ and b that maximize the margin:

$$\max_{\mathbf{w},b} \frac{1}{||\mathbf{w}||} \min_p y_p f(\mathbf{x_p}).$$

Solving this problem is hard as written.

Friday, March 23rd

Definition 3.3. We define the margin as:

$$\begin{aligned} \alpha &= \min_p \frac{y_p f(\mathbf{x_p})}{||\mathbf{w}||} \\ &= \frac{1}{||\mathbf{w}||} \min_p y_p f(\mathbf{x_p}). \end{aligned} \tag{3.13}$$

What you have to note is the decision hyperplane formula $f(\mathbf{x})$ is not unique. Recall $f(\mathbf{x}) = \mathbf{w}^T x + b$. If we define $\hat{f}(x\mathbf{x}) = a\mathbf{w}^T x + ab, a > 0$ will give you the same hyperplane. So there is some redundancy in the way we define our hyperplane. And the margin is unchanged if we scale our hyperplane. So we choose $a = \frac{1}{min_p y_p f(\mathbf{x_p})}$, such that:

$$\min_p y_p f'(\mathbf{x_p}) = \min_p a y_p f(\mathbf{x_p}) = a \min_p y_p f(\mathbf{x_p}) = \frac{\min_p y_p f(\mathbf{x_p})}{\min_p y_p f(\mathbf{x_p})} = 1.$$

Thus for all points we have $y_p f'(\mathbf{x_p}) \geq 1$, since this is the min.

Then the maximum margin optimization becomes:

$$\max_{\mathbf{w},b} \frac{1}{||\mathbf{w}||} \min_p y_p f(\mathbf{x_p}) = \max_{\mathbf{w}',b'} \frac{1}{||\mathbf{w}||} 1.$$

Example 3.4. Minimize $f(\mathbf{x}) = x_1^2 + x_2^2$, subject to $g(\mathbf{x}) = x_1 + x_2 - 1 = 0$. Typically when we add a constraint, our solution changes. Without the constraint, we can just use zero. But it will be nonzero with it. So the way to solve it would be to solve $g(\mathbf{x})$ for one of the variables and then plug it back in, and then take a derivative. We leave it as an exercise to verify that $x_2 = \frac{1}{2} = x_1$ for the solution, and the minimum value is $\frac{1}{2}$.

So the Lagrangian is:

$$\begin{aligned} L(x, \lambda) &= f(x) + \lambda g(x) \\ &= x_1^2 + x_2^2 + \lambda(x_1 + x_2 - 1). \end{aligned} \tag{3.14}$$

Conditions for L to be stationary are:

$$\begin{aligned}\frac{\partial L}{\partial x_1} &= 2x_1 + \lambda = 0 \\ \frac{\partial L}{\partial x_2} &= 2x_1 + \lambda = 0 \\ \frac{\partial L}{\partial \lambda} &= x_1 + x_2 - 1 = 0\end{aligned} \tag{3.15}$$

Then we can solve to find $\lambda = -1$, $x_1 = x_2 = \frac{1}{2}$ which matches the solution we found earlier.

When the equations are more complicated, we must use Lagrange multipliers.

Our primal problem is given by $\min_x f(x)$ such that $g(x) = 0$, our constraint.

Definition 3.5. Lagrangian of the problem:

$$L(x, \lambda) = f(x) + \lambda g(x).$$

We proved earlier that the gradients of f, g must be orthogonal to the tangent plane to the surface g at all times. Then we must have:

$$\nabla_x L = \nabla_x f + \lambda \nabla_x g = 0,$$

which means they must be parallel/(antiparallel).

So we just write the Lagrangian and set the gradient equal to zero.

But recall that the constraint we want to work with is an **inequality** constraint. So now we are working with the surface $g(x) = 0$ as well as it's interior, so we have $g(x) \leq 0$. We are not allowed to be inside our n-dimensional solid as well. In one case, the solution will appear on the surface, in another case, it will appear in the interior. Let x^* denote our solution.

Case 1: If $g(x^*) < 0$ this is called an **inactive constraint.** If this happens, then the solution must be some kind of local minimum with or without the constraint. Thus we know $\nabla f = 0$.

Case 2: If $g(x^*) = 0$ this is called an **active constraint**, and it can be solved in precisely the same way as we did before when we were only dealing with the surface.

In reality, when we solve a problem like this, we are going to assume the solution is on the surface and try to solve it, and then assume the solution is in the interior, and try to solve it, and see which one gives us the correct answer. It's possible they both give optimal solutions.

Definition 3.6. To summarize, the solution of (x^*, λ^*) of $\min_x f(x)$ such that $g(x) \leq 0$ must satisfy:

(1) $\nabla_x L(x^*, \lambda^*) = \nabla f + \lambda \nabla g = 0$: stationary with respect to x.
(2) $g(x^*) \leq 0$: primal feasibility (?)
(3) $\lambda^* \geq 0$: dual feasibility (?)
(4) $\lambda^* g(x^*)0$: complementary slackness (?)

These are known as **KKT conditions** which are necessary and sufficient for a solution to a constrained optimization problem. But it is only sufficient for convex functions.

Remark 3.7. If the solution does not touch the boundary, we must have $\lambda = 0$ (I think).

What about multiple constraints? How about M equality constraints and N inequality constraints? We do the same thing, but with sums. It's in the slides. The KKT conditions are very similar and still work and the problem is pretty easy. There is one Lagrangian variable for each constraint. We don't care about the sign of the variables that correspond to equality constraints, but for inequality constraints, they must be non-negative.

$$L(\mathbf{x}, \lambda, \nu) = f_0(\mathbf{x}) + \sum_i \lambda_i g_i(\mathbf{x}) + \sum_j \nu_j h_j(\mathbf{x})$$

We claim that minimizing the original problem is equivalent to min-max optimization:

$$\min_x \left[\sup_{\lambda \geq 0, \nu} \mathcal{L}(x, \lambda, \nu) \right]$$

Note that the notation for what we are taking the supremum of is just noting that λ are the multipliers for the inequality constraints and so they must be non-negative, and we have no such constraints for ν.

DEFINITION 3.8. We define the dual problem, which is equivalent for convex functions:

$$\max_{\lambda\geq 0,\nu}\left[\inf_x \mathcal{L}(x,\lambda,\nu)\right]$$

Wednesday, March 28th

There will be a second midterm on the 6th of April. Homework 3 will be due in one week on April 4th. There is only one day between the review session and the midterm. This midterm is basically your final exam. The scope of the exam is from Machine Learning onward, so you don't need to know Hidden Markov models. Part of optimization, just a bit. We need to know neural networks as well. We will start presentations from the 13th of April. You should get ready. The order of presentations will be in order of group number. So we will go 9th.

Recall that when you have a constraint optimization, we can use Lagrange multipliers. The Lagrange equation is given by:

$$L(\mathbf{x},\lambda) = f(x) + \lambda g(\mathbf{x}).$$

We are interested in constraints with inequalities, though, not just equalities. We have two kinds of solutions, above.

Technically KKT are only necessary conditions, but when you have a convex problem, it is also sufficient. We talked about **duality**. Recall that we learned finding a solution to the original/primal problem is the same as finding minimizing the supremum of the Lagrangian equation:

$$\min_x\left[\sup_{\lambda\geq 0,\nu} \mathcal{L}(x,\lambda,\nu)\right]. \tag{3.16}$$

The dual problem switches the operations:

$$\max_{\lambda\geq 0,\nu}\left[\inf_x \mathcal{L}(x,\lambda,\nu)\right]. \tag{3.17}$$

REMARK 3.9. Sometimes, dual problems are easier to solve.

EXAMPLE 3.10. Linear Programming:

The primal problem is: $\min_x c^T x$ s.t. $a_i^T x \leq b_i$, for $i = 1, 2, ..., M$.

Recall the Lagrangian is:

$$L(x,\lambda_1,...,\lambda_M) = c^T x + \sum_i \lambda_i(a_i^T x - b_i). \tag{3.18}$$

Dual function: We have: $g(\lambda)\inf_x L(x,\lambda) = \inf_x (c + \sum_i a_i^T \lambda_i)^T x - \sum_i b_i\lambda_i$. The minimum of this linear function will go to infinity if the thing in the parentheses in the first term is not zero, since opponent can choose x s.t. it goes to $-\infty$. So $(c+\sum_i a_i^T\lambda_i)$ must be zero. Otherwise the whole thing is $-\infty$. And so what is left is $-\sum_i b_i\lambda_i$. So the dual problem is:

$$\max_\lambda -b^T\lambda,$$

s.t. $c + A^T\lambda = 0$, $\lambda \geq 0$. The reason you change it to dual problem is it gives you a different interpretation and sometimes it's easier.

EXAMPLE 3.11. Quadratic programming:

We minimize: $\min_{x,y}(x-2)^2 + 2(y-1)^2$ s.t. $x + 4y \leq 3$, and $-x + y \leq 0$.

First thing we do is **write down the Lagrangian**:

$$L(x,y,\lambda_1,\lambda_2) = (x-2)^2 + 2(y-1)^2 + \lambda_1(x+4y-3) + \lambda_2(-x+y). \tag{3.19}$$

Complementary slackness gives us four possible solutions. Which one satisfies the KKT conditions.

These two examples are as difficult as it's going to get for the exam, but we might not even have one on the midterm.

REMARK 3.12. KKT conditions do not guarantee you a unique solution!

We return to support vector machines. What was the original problem? There can be infinitely many solutions that perfectly separate the two classes when the problem is linearly separable. We found that geometrically,

$$\min_p y_p f(\mathbf{x_p})/||\mathbf{w}|| \tag{3.20}$$

is the margin, and we want to maximize it. We managed to change it into the problem:

$$\min_{w,b} \frac{1}{2}||\mathbf{w}||^2,$$

subject to $y_p(\mathbf{w}^T\mathbf{x_p} + b) \geq 1, \forall p$, for linearly separable cases. How do we solve this problem, if you have a good solver that can solve this problem directly, you can use it. It is a linearly constrainted qudaratic problem. So what's the first step?

Write the Langrangian:

Definition 3.13. Lagrangian multipliers:

$$\alpha = (a_1, ..., a_N), a_p \geq 0, \forall p.$$

Definition 3.14. Lagrangian:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{1}||\mathbf{w}||^2 - \sum_p a_p(y_p(\mathbf{w}^T\mathbf{x_p} + b) - 1). \tag{3.21}$$

So we want to set the derivative of this with resepect to $\mathbf{w}$ and b equal to zero. Before knowing what dual functions and stuff are, we were still able to solve the problem by setting it equal to zero and checking KKT conditions, the dual problem just gives us insight into why we are doing this, taking partials and setting equal to zero is minimizing in the dual problem. We have:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \alpha) = 0 = \mathbf{w} - \sum_p a_p y_p \mathbf{x}_p \\ \mathbf{w} = \sum_p a_p y_p \mathbf{x}_p \end{aligned} \tag{3.22}$$

And we do the same for the derivative with respect to b. Plugging back in the values we got for $\mathbf{w}$ and b to Lagrangian, we get:

$$\begin{aligned} L(\mathbf{w}, b\alpha) &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_p a_p(y_p(\mathbf{w}^T\mathbf{x_p} + b) - 1) \\ &= \textit{something} \end{aligned} \tag{3.23}$$

Note the $y_p, y_q, \mathbf{x}_p, \mathbf{x}_q$ are all constants since they are given data.

When D, the dimension of $\mathbf{w}$ is larger than N, the dimension of α, then the dual is faster to solve.

The reason dual is interesting is that we always get the $\mathbf{x}$'s as inner products, so we can replace with kernel function.

On slide 25, we want to get rid of b somehow. I dunno why.

If α_p is zero, then stuff that is not your training points don't affect your decision boundary at all. Those points whose α_p are nonzero are the only points whose information contributes to the definition of $\mathbf{w}$. Only the support vectors affect it. For points whose α_p is nonzero, they must be support vectors. We find b as in the slides and then plug back in to find the full decision function $f(x)$. Whenever you solve a dual problem instead of a primal problem, you must go back to the prime optimal value for the primal problem. We need to check complementary slackness to do this. It will be a small fraction of your training points that become support vectors. And we have to deal with just one other case.

What if it's **non linearly-separable**? There is no straight line that will give you zero training error.

So we'll allow violation, but minimize it. Some points are going to violate $y_p(\mathbf{w}^T\mathbf{x}_p + b) \geq 1$, so instead we write: $y_p(\mathbf{w}^T\mathbf{x}_p + b) \geq 1 - \xi_p$ for $p = 1, ..., N$, where $\xi_1, ..., \xi_N$ are the slack variables. So we solve:

$$\min_{\mathbf{w}, b, \{\ \xi_p \geq 0\ \}} \frac{1}{N}\sum_p \xi_p + \frac{\lambda}{2}||\mathbf{w}||^2.$$

So this other term $\sum_p \xi_p$ is new, used to minimize slackness, minimize violation, if we don't take this, we could just take very large values for ξ_p. So write the Lagrangian and follow the steps, set it equal to zero, and continue all the way to writing the dual function, and this will be a **homework problem**.

There is still another way of solving this which is more practical. It's also equivalent to the following unconstrained problem:

$$\min_{\mathbf{w},b} \frac{1}{N} \sum_p \max \left\{ 0, 1 - y_p(\mathbf{w}^T \mathbf{x}_p + b) \right\} + \frac{\lambda}{2} ||\mathbf{w}||^2. \tag{3.24}$$

Definition 3.15. Non linearly-separable problems are called **soft-margin SVM** problems.

The opposite are called hard-margin SVM. Usually, we're solving soft-margin, we almost never have a hard margin, so the term SVM usually refers to soft-margin.

For an unconstrained optimization problem, we just compute the gradient and do gradient on this alternate form defined above. It isn't that difficult even though we have this max function inside. We have a cost function for our alternative form. Look at it in cases, when the two quantities we are taking the max of, 0 and some expression, well we consider when the expression is nonnegative and when it is negative. And the max is trivial when it is negative. So you just check whether it is or not, and then solve normally.

If you are trying to use SVM for your project, there are different solvers.

Friday, March 30th

On the homework, one of the problems will ask you to rederive the solution of SVM. Another problem will ask you to do the same for the Soft-Margin SVM. The process is almost the same, and it won't be particularly difficult.

What made SVM so popular? It's relatively easier to solve. It has an intuitive geometric motivation. Optimization was numerically easy. It can be kernelized to nonlinear problems. You replace all the dot products with some function of the two vectors, nonlinear version of inner product.

3.4. Neural Networks

Think about a biological neuron. It is partially true that neural networks are inspired by biological neuron. It has dendrites to receive information from other neurons, these signals are accumulated, it fires along the axon, and the signal stimulates other neurons.

In our neural network units, we have dendrites with coefficients which determine strength of signal. Total signal is passed through threshold function, and if it is greater than a threshold, neuron fires a nonzero value into other neurons. We also have a bias dendrite. This is an artificial neural network made of artificial neurons.

If you have n neurons in the system, we will have somewhere around n^2 connections. We can use a hard threshold, think of a softmax but with corners. Or we could use softmax, which is more like logistic regression, and is exactly that with a single neuron. It is a linear classifier. We can also use hyperbolic tangent. This has the same shape but it's centered at origin in $\mathbb{R}^2$, where softmax is centered at $(0, 1/2)$.

Definition 3.16. A neural net is also called a **perceptron**, this is an antiquated term from the 80's.

To implement or, try 0.5 on bias, 1 on the other two.

So what is **forward propagation**? Say a_1, a_2, a_3 are the activation of the input layer at the bottom. The output layer at the top are just the same as the outputs. The middle layers are a_4, a_5, a_6. Note $a_4 = g(n_4) = g(\sum_j w_{ij} a_j)$.

It's possible to use no thresholding function in outer layer. We have $f_i = a_i = n_i = \sum_k W_{ki} a_k$. We could also do a softmax in this layer, or we could use a normal thresholding function.

3.4.1. Matrix calculus.

$$\begin{aligned} \nabla_x x^T A x &= 2Ax. \\ \nabla_x b^T x &= b. \end{aligned} \tag{3.25}$$

3.5. Exam II Review

'