Example

Let's start with an example. Suppose we are charged with providing automated access control to a building. Before entering the building each person has to look into a camera so we can take a still image of their face. For our purposes it suffices just to decide based on the image whether the person can enter the building. It might be helpful to (try to) also identify each person but this might require type of information we do not have (e.g., names or whether any two face images correspond to the same person). We only have face images of people recorded while access control was still provided manually. As a result of this experience we have *labeled* images. An image is labeled *positive* if the person in question should gain entry and *negative* otherwise. To supplement the set of negatively labeled images (as we would expect only few cases of refused entries under normal circumstances) we can use any other face images of people who we do not expect to be permitted to enter the building. Images taken with similar camera-face orientation (e.g., from systems operational in other buildings) would be preferred. Our task then is to come up with a function – a *classifier* – that maps pixel images to binary (± 1) labels. And we only have the small set of labeled images (the *training set*) to constrain the function.

Let's make the task a bit more formal. We assume that each image (grayscale) is represented as a column vector $\mathbf{x}$ of dimension d. So, the pixel intensity values in the image, column by column, are concatenated into a single column vector. If the image has 100 by 100 pixels, then $d = 10000$. We assume that all the images are of the same size. Our classifier is a binary valued function $f : \mathcal{R}^d \to \{-1, 1\}$ chosen on the basis of the training set alone. For our task here we assume that the classifier knows nothing about images (or faces for that matter) beyond the labeled training set. So, for example, from the point of view of the classifier, the images could have been measurements of weight, height, etc. rather than pixel intensities. The classifier only has a set of n training vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ with binary ± 1 labels $y_1, \ldots, y_n$. This is the only information about the task that we can use to constraint what the function f should be.

What kind of solution would suffice?

Suppose now that we have $n = 50$ labeled pixel images that are 128 by 128, and the pixel intensities range from 0 to 255. It is therefore possible that we can find a single pixel, say pixel i, such that each of our n images have a distinct value for that pixel. We could then construct a simple binary function based on this single pixel that perfectly maps the training images to their labels. In other words, if x_{ti} refers to pixel i in the t^{th} training

image, and x'_i is the i^{th} pixel in any image $\mathbf{x}'$, then

$$f_i(\mathbf{x}') = \begin{cases} y_t, \text{ if } x_{ti} = x'_i \text{ for some } t = 1, \ldots, n \text{ (in this order)} \\ -1, \text{ otherwise} \end{cases} \quad (1)$$

would appear to solve the task. In fact, it is always possible to come up with such a "perfect" binary function if the training images are distinct (no two images have identical pixel intensities for all pixels). But do we expect such rules to be useful for images not in the training set? Even an image of the same person varies somewhat each time the image is taken (orientation is slightly different, lighting conditions may have changed, etc). These rules provide no sensible predictions for images that are not identical to those in the training set. The primary reason for why such trivial rules do not suffice is that our task is *not* to correctly classify the training images. Our task is to find a rule that works well for all new images we would encounter in the access control setting; the training set is merely a helpful source of information to find such a function. To put it a bit more formally, we would like to find classifiers that *generalize* well, i.e., classifiers whose performance on the training set is representative of how well it works for yet unseen images.

Model selection

So how can we find classifiers that generalize well? The key is to constrain the set of possible binary functions we can entertain. In other words, we would like to find a class of binary functions such that if a function in this class works well on the training set, it is also likely to work well on the unseen images. The "right" class of functions to consider cannot be too large in the sense of containing too many clearly different functions. Otherwise we are likely to find rules similar to the trivial ones that are close to perfect on the training set but do not generalize well. The class of function should not be too small either or we run the risk of not finding any functions in the class that work well even on the training set. If they don't work well on the training set, how could we expect them to work well on the new images? Finding the class of functions is a key problem in machine learning, also known as the *model selection* problem.

Linear classifiers through origin

Let's just fix the function class for now. Specifically, we will consider only a type of *linear classifiers*. These are thresholded linear mappings from images to labels. More formally, we only consider functions of the form

$$f(\mathbf{x}; \theta) = \text{sign}\big(\theta_1 x_1 + \ldots + \theta_d x_d\big) = \text{sign}\big(\theta^T \mathbf{x}\big) \quad (2)$$

where $\theta = [\theta_1, \ldots, \theta_d]^T$ is a column vector of real valued parameters. Different settings of the parameters give different functions in this class, i.e., functions whose value or *output* in $\{-1, 1\}$ could be different for some *input* images $\mathbf{x}$. Put another way, the functions in our class are parameterized by $\theta \in \mathcal{R}^d$.

We can also understand these linear classifiers geometrically. The classifier changes its prediction only when the argument to the sign function changes from positive to negative (or vice versa). Geometrically, in the space of image vectors, this transition corresponds to crossing the *decision boundary* where the argument is exactly zero: all $\mathbf{x}$ such that $\theta^T\mathbf{x} = 0$. The equation defines a plane in d-dimensions, a plane that goes through the origin since $\mathbf{x} = 0$ satisfies the equation. The parameter vector θ is normal (orthogonal) to this plane; this is clear since the plane is defined as all $\mathbf{x}$ for which $\theta^T\mathbf{x} = 0$. The θ vector as the normal to the plane also specifies the direction in the image space along which the value of $\theta^T\mathbf{x}$ would increase the most. Figure 1 below tries to illustrate these concepts.

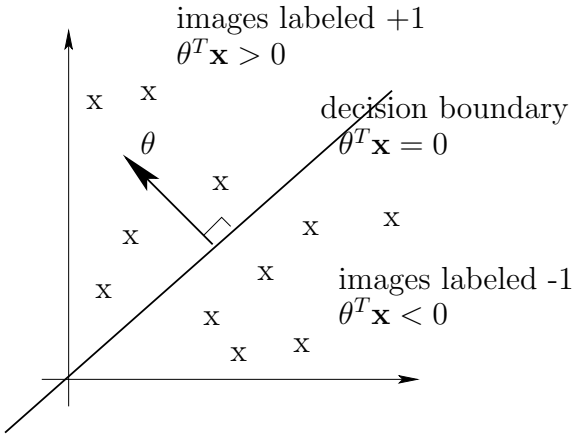

Figure 1: A linear classifier through origin.

Before moving on let's figure out whether we lost some useful properties of images as a result of restricting ourselves to linear classifiers? In fact, we did. Consider, for example, how nearby pixels in face images relate to each other (e.g., continuity of skin). This information is completely lost. The linear classifier is perfectly happy (i.e., its ability to classify images remains unchanged) if we get images where the pixel positions have been reordered provided that we apply the same transformation to all the images. This permutation of pixels merely reorders the terms in the argument to the sign function in Eq. (2). A linear classifier therefore does not have access to information about which pixels are close to each other in the image.

Learning algorithm: the perceptron

Now that we have chosen a function class (perhaps suboptimally) we still have to find a specific function in this class that works well on the training set. This is often referred to as the *estimation* problem. Let's be a bit more precise. We'd like to find a linear classifier that makes the fewest mistakes on the training set. In other words, we'd like to find θ that minimizes the *training error*

$$\hat{E}(\theta) = \frac{1}{n}\sum_{t=1}^{n}\left(1 - \delta(y_t, f(\mathbf{x}_t;\theta))\right) = \frac{1}{n}\sum_{t=1}^{n}\text{Loss}(y_t, f(\mathbf{x}_t;\theta)) \tag{3}$$

where $\delta(y, y') = 1$ if $y = y'$ and 0 otherwise. The training error merely counts the average number of training images where the function predicts a label different from the label provided for that image. More generally, we could compare our predictions to labels in terms of a loss function $\text{Loss}(y_t, f(\mathbf{x}_t;\theta))$. This is useful if errors of a particular kind are more costly than others (e.g., letting a person enter the building when they shouldn't). For simplicity, we use the *zero-one loss* that is 1 for mistakes and 0 otherwise.

What would be a reasonable algorithm for setting the parameters θ? Perhaps we can just incrementally adjust the parameters so as to correct any mistakes that the corresponding classifier makes. Such an algorithm would seem to reduce the training error that counts the mistakes. Perhaps the simplest algorithm of this type is the *perceptron* update rule. We consider each training image one by one, cycling through all the images, and adjust the parameters according to

$$\theta' \leftarrow \theta + y_t\mathbf{x}_t \;\text{ if }\; y_t \neq f(\mathbf{x}_t;\theta) \tag{4}$$

In other words, the parameters (classifier) is changed only if we make a mistake. These updates tend to correct mistakes. To see this, note that when we make a mistake the sign of $\theta^T\mathbf{x}_t$ disagrees with y_t and the product $y_t\theta^T\mathbf{x}_t$ is negative; the product is positive for correctly classified images. Suppose we make a mistake on $\mathbf{x}_t$. Then the updated parameters are given by $\theta' = \theta + y^t\mathbf{x}_t$, written here in a vector form. If we consider classifying the same image $\mathbf{x}_t$ after the update, then

$$y_t\theta'^T\mathbf{x}^t = y_t(\theta + y_t\mathbf{x}_t)^T\mathbf{x}_t = y_t\theta^T\mathbf{x}_t + y_t^2\mathbf{x}_t^T\mathbf{x}_t = y_t\theta^T\mathbf{x}_t + \|\mathbf{x}_t\|^2 \tag{5}$$

In other words, the value of $y_t\theta^T\mathbf{x}_t$ increases as a result of the update (becomes more positive). If we consider the same image repeatedly, then we will necessarily change the parameters such that the image is classified correctly, i.e., the value of $y_t\theta^T\mathbf{x}_t$ becomes positive. Mistakes on other images may steer the parameters in different directions so it may not be clear that the algorithm converges to something useful if we repeatedly cycle through the training images.

Analysis of the perceptron algorithm

The perceptron algorithm ceases to update the parameters only when all the training images are classified correctly (no mistakes, no updates). So, if the training images are possible to classify correctly with a linear classifier, will the perceptron algorithm find such a classifier? Yes, it does, and it will converge to such a classifier in a finite number of updates (mistakes). We'll show this in lecture 2.

Perceptron, convergence, and generalization

Recall that we are dealing with linear classifiers through origin, i.e.,

$$f(\mathbf{x};\theta) = \text{sign}\big(\,\theta^T\mathbf{x}\,\big) \tag{1}$$

where $\theta \in \mathcal{R}^d$ specifies the parameters that we have to estimate on the basis of training examples (images) $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and labels $y_1, \ldots, y_n$.

We will use the perceptron algorithm to solve the estimation task. Let k denote the number of parameter updates we have performed and $\theta^{(k)}$ the parameter vector after k updates. Initially $k = 0$ and $\theta^{(k)} = 0$. The algorithm then cycles through all the training instances $(\mathbf{x}_t, y_t)$ and updates the parameters only in response to mistakes, i.e., when the label is predicted incorrectly. More precisely, we set $\theta^{(k+1)} = \theta^{(k)} + y_t\mathbf{x}_t$ when $y_t(\theta^{(k)})^T\mathbf{x}_t < 0$ (mistake), and otherwise leave the parameters unchanged.

Convergence in a finite number of updates

Let's now show that the perceptron algorithm indeed convergences in a finite number of updates. The same analysis will also help us understand how the linear classifier generalizes to unseen images. To this end, we will assume that all the (training) images have bounded Euclidean norms, i.e., $\|\mathbf{x}_t\| \leq R$ for all t and some finite R. This is clearly the case for any pixel images with bounded intensity values. We also make a much stronger assumption that there exists a linear classifier in our class with finite parameter values that correctly classifies all the (training) images. More precisely, we assume that there is some $\gamma > 0$ such that $y_t(\theta^*)^T\mathbf{x}_t \geq \gamma$ for all $t = 1, \ldots, n$. The additional number $\gamma > 0$ is used to ensure that each example is classified correctly with a *finite margin*.

The convergence proof is based on combining two results: 1) we will show that the inner product $(\theta^*)^T\theta^{(k)}$ increases at least linearly with each update, and 2) the squared norm $\|\theta^{(k)}\|^2$ increases at most linearly in the number of updates k. By combining the two we can show that the cosine of the angle between $\theta^{(k)}$ and θ^* has to increase by a finite increment due to each update. Since cosine is bounded by one, it follows that we can only make a finite number of updates.

Part 1: we simply take the inner product $(\theta^*)^T\theta^{(k)}$ before and after each update. When making the k^{th} update, say due to a mistake on image $\mathbf{x}_t$, we get

$$(\theta^*)^T\theta^{(k)} = (\theta^*)^T\theta^{(k-1)} + y_t(\theta^*)^T\mathbf{x}_t \geq (\theta^*)^T\theta^{(k-1)} + \gamma \tag{2}$$

since, by assumption, $y_t(\theta^*)^T\mathbf{x}_t \geq \gamma$ for all t (θ^* is always correct). Thus, after k updates,

$$(\theta^*)^T\theta^{(k)} \geq k\gamma \tag{3}$$

Part 2: Our second claim follows simply from the fact that updates are made only on mistakes:

$$\begin{aligned}
\|\theta^{(k)}\|^2 &= \|\theta^{(k-1)} + y_t\mathbf{x}_t\|^2 && (4)\\
&= \|\theta^{(k-1)}\|^2 + 2y_t(\theta^{(k-1)})^T\mathbf{x}_t + \|\mathbf{x}_t\|^2 && (5)\\
&\leq \|\theta^{(k-1)}\|^2 + \|\mathbf{x}_t\|^2 && (6)\\
&\leq \|\theta^{(k-1)}\|^2 + R^2 && (7)
\end{aligned}$$

since $y_t(\theta^{(k-1)})^T\mathbf{x}^t < 0$ whenever an update is made and, by assumption, $\|\mathbf{x}_t\| \leq R$. Thus,

$$\|\theta^{(k)}\|^2 \leq kR^2 \tag{8}$$

We can now combine parts 1) and 2) to bound the cosine of the angle between θ^* and $\theta^{(k)}$:

$$cos(\theta^*, \theta^{(k)}) = \frac{(\theta^*)^T\theta^{(k)}}{\|\theta^{(k)}\|\,\|\theta^*\|} \overset{1)}{\geq} \frac{k\gamma}{\|\theta^{(k)}\|\,\|\theta^*\|} \overset{2)}{\geq} \frac{k\gamma}{\sqrt{kR^2}\,\|\theta^*\|} \tag{9}$$

Since cosine is bounded by one, we get

$$1 \geq \frac{k\gamma}{\sqrt{kR^2}\,\|\theta^*\|} \text{ or } k \leq \frac{R^2\|\theta^*\|^2}{\gamma^2} \tag{10}$$

Margin and geometry

It is worthwhile to understand this result a bit further. For example, does $\|\theta^*\|^2/\gamma^2$ relate to how difficult the classification problem is? Indeed, it does. We claim that its inverse, i.e., $\gamma/\|\theta^*\|$ is the smallest distance in the image space from any example (image) to the decision boundary specified by θ^*. In other words, it serves as a measure of how well the two classes of images are separated (by a linear boundary). We will call this the geometric margin or γ_{geom} (see figure 1). γ_{geom}^{-1} is then a fair measure of how difficult the problem is: the smaller the geometric margin that separates the training images, the more difficult the problem.

To calculate γ_{geom} we measure the distance from the decision boundary ${\theta^*}^T\mathbf{x} = 0$ to one of the images $\mathbf{x}_t$ for which $y_t{\theta^*}^T\mathbf{x}_t = \gamma$. Since θ^* specifies the normal to the decision boundary,

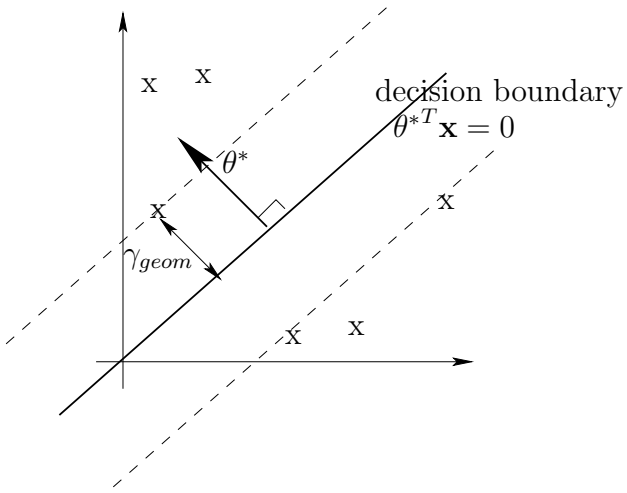

Figure 1: Geometric margin

the shortest path from the boundary to the image $\mathbf{x}_t$ will be parallel to the normal. The image for which $y_t \theta^{*T} \mathbf{x}_t = \gamma$ is therefore among those closest to the boundary. Now, let's define a line segment from $\mathbf{x}(0) = \mathbf{x}_t$, parallel to θ^*, towards the boundary. This is given by

$$\mathbf{x}(\xi) = \mathbf{x}(0) - \xi \frac{y_t \theta^*}{\|\theta^*\|} \tag{11}$$

where ξ defines the length of the line segment since it multiplies a unit length vector. It remains to find the value of ξ such that $\theta^{*T}\mathbf{x}(\xi) = 0$, or, equivalently, $y_t \theta^{*T}\mathbf{x}(\xi) = 0$. This is the point where the segment hits the decision boundary. Thus

$$\begin{aligned} y_t \theta^{*T}\mathbf{x}(\xi) &= y_t \theta^{*T}\left[\mathbf{x}(0) - \xi \frac{y_t \theta^*}{\|\theta^*\|}\right] && (12)\\ &= y_t \theta^{*T}\left[\mathbf{x}_t - \xi \frac{y_t \theta^*}{\|\theta^*\|}\right] && (13)\\ &= y_t \theta^{*T}\mathbf{x}_t - \xi \frac{\|\theta^*\|^2}{\|\theta^*\|} && (14)\\ &= \gamma - \xi\|\theta^*\| = 0 && (15) \end{aligned}$$

implying that the distance is exactly $\xi = \gamma/\|\theta^*\|$ as claimed. As a result, the bound on the number of perceptron updates can be written more succinctly in terms of the geometric

margin γ_{geom} (distance to the boundary):

$$k \leq \left(\frac{R}{\gamma_{geom}} \right)^2 \tag{16}$$

with the understanding that γ_{geom} is the largest geometric margin that could be achieved by a linear classifier for this problem. Note that the result does not depend (directly) on the dimension d of the examples, nor the number of training examples n. It is nevertheless tempting to interpret $\left(\frac{R}{\gamma_{geom}} \right)^2$ as a measure of difficulty (or complexity) of the problem of learning linear classifiers in this setting. You will see later in the course that this is exactly the case, cast in terms of a measure known as *VC-dimension*.

Generalization guarantees

We have so far discussed the perceptron algorithm only in relation to the training set but we are more interested in how well the perceptron classifies images we have not yet seen, i.e., how well it generalizes to new images. Our simple analysis above actually provides some information about generalization. Let's assume then that all the images and labels we could possibly encounter satisfy the same two assumptions. In other words, 1) $\|\mathbf{x}_t\| \leq R$ and 2) $y_t \theta^{*T} \mathbf{x}_t \geq \gamma$ for all t and some finite θ^*. So, in essence, we assume that there is a linear classifier that works for all images and labels in this problem, we just don't know what this linear classifier is to start with. Let's now imagine getting the images and labels one by one and performing only a single update per image, if misclassified, and move on. The previous situation concerning the training set corresponds to encountering the same set of images repeatedly. How many mistakes are we now going to make in this infinite arbitrary sequence of images and labels, subject only to the two assumptions? The same number $k \leq (R/\gamma_{geom})^2$. Once we have made this many mistakes we would classify all the new images correctly. So, provided that the two assumptions hold, especially the second one, we obtain a nice guarantee of generalization. One caveat here is that the perceptron algorithm does need to know when it has made a mistake. The bound is after all cast in terms of the number of updates based on mistakes.

Maximum margin classifier?

We have so far used a simple on-line algorithm, the perceptron algorithm, to estimate a linear classifier. Our reference assumption has been, however, that there exists a linear classifier that has a large geometric margin, i.e., whose decision boundary is well separated

from all the training images (examples). Can't we find such a large margin classifier directly? Yes, we can. The classifier is known as the Support Vector Machine or SVM for short. See the next lecture for details.

The Support Vector Machine

So far we have used a reference assumption that there exists a linear classifier that has a large geometric margin, i.e., whose decision boundary is well separated from all the training images (examples). Such a large margin classifier seems like one we would like to use. Can't we find it more directly? Yes, we can. The classifier is known as the Support Vector Machine or SVM for short.

You could imagine finding the maximum margin linear classifier by first identifying any classifier that correctly classifies all the examples (Figure 2a) and then increasing the geometric margin until the classifier "locks in place" at the point where we cannot increase the margin any further (Figure 2b). The solution is unique.

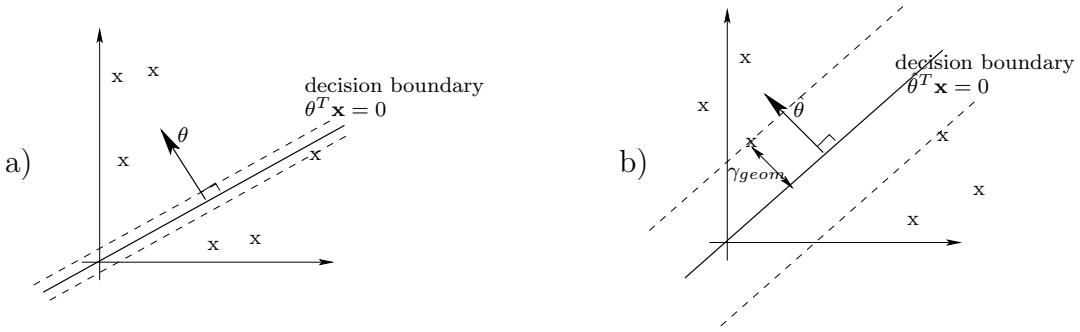

Figure 1: a) A linear classifier with a small geometric margin, b) maximum margin linear classifier.

More formally, we can set up an optimization problem for directly maximizing the geometric margin. We will need the classifier to be correct on all the training examples or $y_t \theta^T \mathbf{x}_t \geq \gamma$ for all $t = 1, \ldots, n$. Subject to these constraints, we would like to maximize $\gamma / \|\theta\|$, i.e., the geometric margin. We can alternatively minimize the inverse $\|\theta\|/\gamma$ or the inverse squared $\frac{1}{2}(\|\theta\|/\gamma)^2$ subject to the same constraints (the factor 1/2 is included merely for later convenience). We then have the following optimization problem for finding $\hat{\theta}$:

$$\text{minimize } \frac{1}{2}\|\theta\|^2/\gamma^2 \text{ subject to } y_t \theta^T \mathbf{x}_t \geq \gamma \text{ for all } t = 1, \ldots, n \tag{1}$$

We can simplify this a bit further by getting rid of γ. Let's first rewrite the optimization problem in a way that highlights how the solution depends (or doesn't depend) on γ:

$$\text{minimize } \frac{1}{2}\|\theta/\gamma\|^2 \text{ subject to } y_t (\theta/\gamma)^T \mathbf{x}_t \geq 1 \text{ for all } t = 1, \ldots, n \tag{2}$$

In other words, our classification problem (the data, setup) only tells us something about the ratio θ/γ, not θ or γ separately. For example, the geometric margin is defined only on the basis of this ratio. Scaling θ by a constant also doesn't change the decision boundary. We can therefore fix $\gamma = 1$ and solve for θ from

$$\text{minimize } \frac{1}{2}\|\theta\|^2 \text{ subject to } y_t\theta^T\mathbf{x}_t \geq 1 \text{ for all } t = 1, \ldots, n \tag{3}$$

This optimization problem is in the standard SVM form and is a *quadratic programming* problem (objective is quadratic in the parameters with linear constraints). The resulting geometric margin is $1/\|\hat{\theta}\|$ where $\hat{\theta}$ is the unique solution to the above problem. The decision boundary (separating hyper-plane) nor the value of the geometric margin were affected by our choice $\gamma = 1$.

General formulation, offset parameter

We will modify the linear classifier here slightly by adding an offset term so that the decision boundary does not have to go through the origin. In other words, the classifier that we consider has the form

$$f(\mathbf{x}; \theta, \theta_0) = \text{sign}(\, \theta^T\mathbf{x} + \theta_0 \,) \tag{4}$$

with parameters θ (normal to the separating hyper-plane) and the offset parameter θ_0, a real number. As before, the equation for the separating hyper-plane is obtained by setting the argument to the sign function to zero or $\theta^T\mathbf{x} + \theta_0 = 0$. This is a general equation for a hyper-plane (line in 2-dim). The additional offset parameter can lead to a classifier with a larger geometric margin. This is illustrated in Figures 2a and 2b. Note that the vector $\hat{\theta}$ corresponding to the maximum margin solution is different in the two figures.

The offset parameter changes the optimization problem only slightly:

$$\text{minimize } \frac{1}{2}\|\theta\|^2 \text{ subject to } y_t(\theta^T\mathbf{x}_t + \theta_0) \geq 1 \text{ for all } t = 1, \ldots, n \tag{5}$$

Note that the offset parameter only appears in the constraints. This is different from simply modifying the linear classifier through origin by feeding it with examples that have an additional constant component, i.e., $\mathbf{x}' = [\mathbf{x}; 1]$. In the above formulation we do not bias in any way where the separating hyper-plane should appear, only that it should maximize the geometric margin.

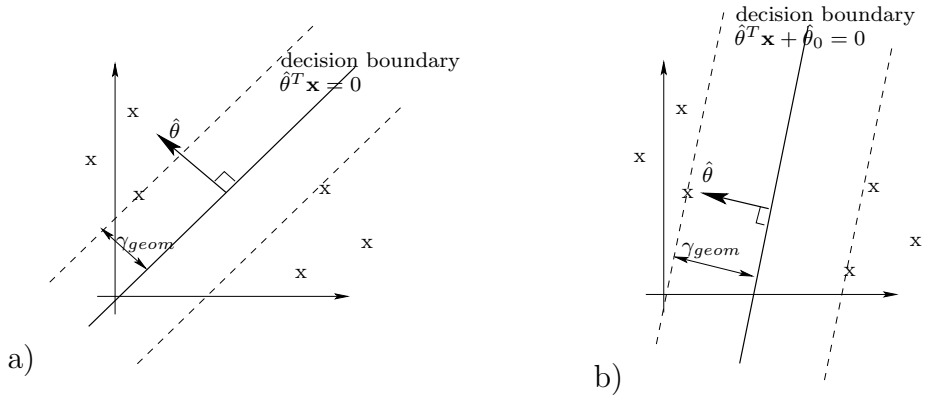

Figure 2: a) Maximum margin linear classifier through origin; b) Maximum margin linear classifier with an offset parameter

Properties of the maximum margin linear classifier

The maximum margin classifier has several very nice properties, and some not so advantageous features.

Benefits. On the positive side, we have already motivated these classifiers based on the perceptron algorithm as the "best reference classifiers". The solution is also unique based on any linearly separable training set. Moreover, drawing the separating boundary as far from the training examples as possible makes it robust to noisy examples (though not noisy labels). The maximum margin linear boundary also has the curious property that the solution depends only on a subset of the examples, those that appear exactly on the margin (the dashed lines parallel to the boundary in the figures). The examples that lie exactly on the margin are called *support vectors* (see Figure 3). The rest of the examples could lie anywhere outside the margin without affecting the solution. We would therefore get the same classifier if we had only received the support vectors as training examples. Is this is a good thing? To answer this question we need a bit more formal (and fair) way of measuring how good a classifier is.

One possible "fair" performance measure evaluated only on the basis of the training examples is *cross-validation*. This is simply a method of retraining the classifier with subsets of training examples and testing it on the remaining held-out (and therefore fair) examples, pretending we had not seen them before. A particular version of this type of procedure is called *leave-one-out cross-validation*. As the name suggests, the procedure is defined as follows: select each training example in turn as the single example to be held-out, train the

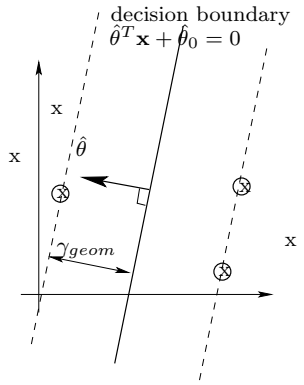

Figure 3: Support vectors (circled) associated the maximum margin linear classifier

classifier on the basis of all the remaining training examples, test the resulting classifier on the held-out example, and count the errors. More precisely, let the superscript '$-i$' denote the parameters we would obtain by finding the maximum margin linear separator without the i^{th} training example. Then

$$\text{leave-one-out CV error} = \frac{1}{n}\sum_{i=1}^{n} \text{Loss}\Big(y_i, f(\mathbf{x}_i; \theta^{-i}, \theta_0^{-i})\Big) \tag{6}$$

where $\text{Loss}(y, y')$ is the zero-one loss. We are effectively trying to gauge how well the classifier would generalize to each training example if it had not been part of the training set. A classifier that has a low leave-one-out cross-validation error is likely to generalize well though it is not guaranteed to do so.

Now, what is the leave-one-out CV error of the maximum margin linear classifier? Well, examples that lie outside the margin would be classified correctly regardless of whether they are part of the training set. Not so for support vectors. They are key to defining the linear separator and thus, if removed from the training set, may be misclassified as a result. We can therefore derive a simple upper bound on the leave-one-out CV error:

$$\text{leave-one-out CV error} \leq \frac{\#\text{ of support vectors}}{n} \tag{7}$$

A small number of support vectors – a sparse solution – is therefore advantageous. This is another argument in favor of the maximum margin linear separator.

Problems. There are problems as well, however. Even a single training example, if mislabeled, can radically change the maximum margin linear classifier. Consider, for example,

what would happen if we switched the label of the top right support vector in Figure 3.

Allowing misclassified examples, relaxation

Labeling errors are common in many practical problems and we should try to mitigate their effect. We typically do not know whether examples are difficult to classify because of labeling errors or because they simply are not linearly separable (there isn't a linear classifier that can classify them correctly). In either case we have to articulate a trade-off between misclassifying a training example and the potential benefit for other examples.

Perhaps the simplest way to permit errors in the maximum margin linear classifier is to introduce "slack" variables for the classification/margin constraints in the optimization problem. In other words, we measure the degree to which each margin constraint is violated and associate a cost for the violation. The costs of violating constraints are minimized together with the norm of the parameter vector. This gives rise to a simple relaxed optimization problem:

$$\text{minimize } \frac{1}{2}\|\theta\|^2 + C\sum_{t=1}^{n}\xi_t \tag{8}$$

$$\text{subject to } \; y_t(\theta^T\mathbf{x}_t + \theta_0) \geq 1 - \xi_t \;\text{ and } \xi_t \geq 0 \;\text{ for all } t = 1, \ldots, n \tag{9}$$

where ξ_t are the slack variables. The margin constraint is violated when we have to set $\xi_t > 0$ for some example. The penalty for this violation is $C\xi_t$ and it is traded-off with the possible gain in minimizing the squared norm of the parameter vector or $\|\theta\|^2$. If we increase the penalty C for margin violations then at some point all $\xi_t = 0$ and we get back the maximum margin linear separator (if possible). On the other hand, for small C many margin constraints can be violated. Note that the relaxed optimization problem specifies a particular *quantitative* trade-off between the norm of the parameter vector and margin violations. It is reasonable to ask whether this is indeed the trade-off we want.

Let's try to understand the setup a little further. For example, what is the resulting margin when some of the constraints are violated? We can still take $1/\|\hat{\theta}\|$ as the geometric margin. This is indeed the geometric margin based on examples for which $\xi_t^* = 0$ where '*' indicates the optimized value. So, is it the case that we get the maximum margin linear classifier for the subset of examples for which $\xi_t^* = 0$? No, we don't. The examples that violate the margin constraints, including those training examples that are actually misclassified (larger violation), do affect the solution. In other words, the parameter vector $\hat{\theta}$ is defined on the basis of examples right at the margin, those that violate the constraint but not enough to

be misclassified, and those that are misclassified. All of these are support vectors in this sense.

The Support Vector Machine and regularization

We proposed a simple relaxed optimization problem for finding the maximum margin separator when some of the examples may be misclassified:

$$\text{minimize } \frac{1}{2}\|\theta\|^2 + C\sum_{t=1}^{n}\xi_t \tag{1}$$

$$\text{subject to } \; y_t(\theta^T\mathbf{x}_t + \theta_0) \geq 1 - \xi_t \;\text{ and } \xi_t \geq 0 \;\text{ for all } t = 1, \ldots, n \tag{2}$$

where the remaining parameter C could be set by cross-validation, i.e., by minimizing the leave-one-out cross-validation error.

The goal here is to briefly understand the relaxed optimization problem from the point of view of *regularization*. Regularization problems are typically formulated as optimization problems involving the desired objective (classification loss in our case) and a regularization penalty. The regularization penalty is used to help stabilize the minimization of the objective or infuse prior knowledge we might have about desirable solutions. Many machine learning methods can be viewed as regularization methods in this manner. For later utility we will cast SVM optimization problem as a regularization problem.

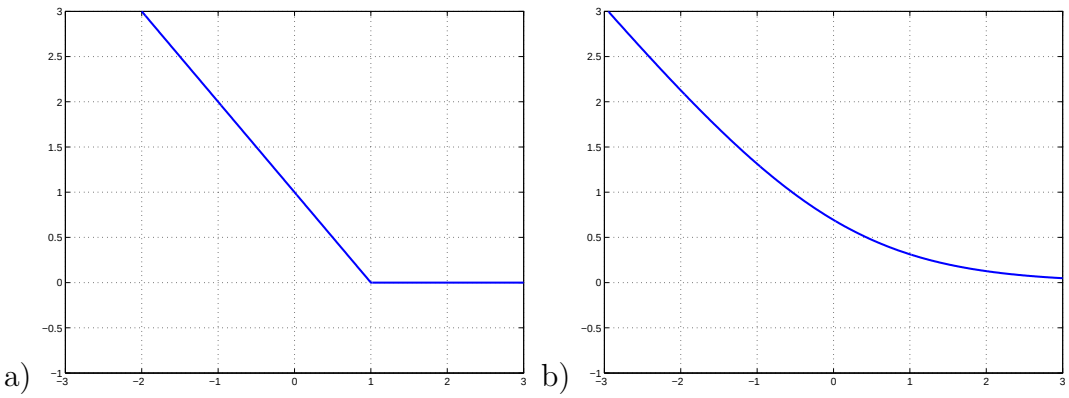

Figure 1: a) The hinge loss $(1-z)^+$ as a function of z. b) The logistic loss $\log[1+\exp(-z)]$ as a function of z.

To turn the relaxed optimization problem into a regularization problem we define a loss function that corresponds to individually optimized ξ_t values and specifies the cost of violating each of the margin constraints. We are effectively solving the optimization problem with respect to the ξ values for a fixed θ and θ_0. This will lead to an expression of $C\sum_t \xi_t$ as a function of θ and θ_0.

The loss function we need for this purpose is based on the *hinge loss* $\text{Loss}_h(z)$ defined as the positive part of $1 - z$, written as $(1 - z)^+$ (see Figure 1a). The relaxed optimization problem can be written using the hinge loss as

$$\text{minimize } \frac{1}{2}\|\theta\|^2 + C\sum_{t=1}^{n} \overbrace{\left(1 - y_t(\theta^T \mathbf{x}_t + \theta_0)\right)^+}^{=\hat{\xi}_t} \tag{3}$$

Here $\|\theta\|^2/2$, the inverse squared geometric margin, is viewed as a regularization penalty that helps stabilize the objective

$$C\sum_{t=1}^{n} \left(1 - y_t(\theta^T \mathbf{x}_t + \theta_0)\right)^+ \tag{4}$$

In other words, when no margin constraints are violated (zero loss), the regularization penalty helps us select the solution with the largest geometric margin.

Logistic regression, maximum likelihood estimation

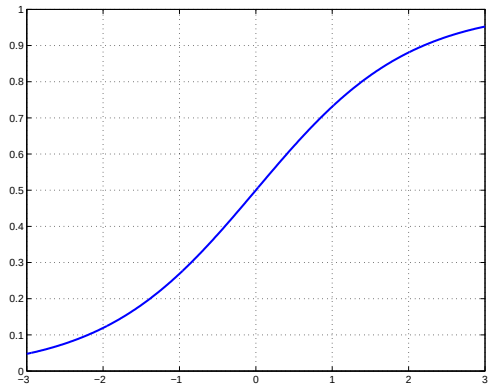

Figure 2: The logistic function $g(z) = (1 + \exp(-z))^{-1}$.

Another way of dealing with noisy labels in linear classification is to model how the noisy labels are generated. For example, human assigned labels tend to be very good for "typical examples" but exhibit some variation in more difficult cases. One simple model of noisy labels in linear classification is a logistic regression model. In this model we assign a

probability distribution over the two labels in such a way that the labels for examples further away from the decision boundary are more likely to be correct. More precisely, we say that

$$P(y = 1|\mathbf{x}, \theta, \theta_0) = g\left(\theta^T\mathbf{x} + \theta_0\right) \tag{5}$$

where $g(z) = (1 + \exp(-z))^{-1}$ is known as the logistic function (Figure 2). One way to derive the form of the logistic function is to say that the *log-odds* of the predicted class probabilities should be a linear function of the inputs:

$$\log \frac{P(y = 1|\mathbf{x}, \theta, \theta_0)}{P(y = -1|\mathbf{x}, \theta, \theta_0)} = \theta^T\mathbf{x} + \theta_0 \tag{6}$$

So for example, when we predict the same probability (1/2) for both classes, the log-odds term is zero and we recover the decision boundary $\theta^T\mathbf{x} + \theta_0 = 0$. The precise functional form of the logistic function, or, equivalently, the fact that we chose to model log-odds with the linear prediction, may seem a little arbitrary (but perhaps not more so than the hinge loss used with the SVM classifier). We will derive the form of the logistic function later on in the course based on certain assumptions about *class-conditional* distributions $P(\mathbf{x}|y = 1)$ and $P(\mathbf{x}|y = -1)$.

In order to better compare the logistic regression model with the SVM we will write the conditional probability $P(y|\mathbf{x}, \theta, \theta_0)$ a bit more succinctly. Specifically, since $1 - g(z) = g(-z)$ we get

$$P(y = -1|\mathbf{x}, \theta, \theta_0) = 1 - P(y = 1|\mathbf{x}, \theta, \theta_0) = 1 - g(\,\theta^T\mathbf{x} + \theta_0\,) = g\left(-(\theta^T\mathbf{x} + \theta_0)\right) \tag{7}$$

and therefore

$$P(y|\mathbf{x}, \theta, \theta_0) = g\left(\,y(\theta^T\mathbf{x} + \theta_0)\right) \tag{8}$$

So now we have a linear classifier that makes probabilistic predictions about the labels. How should we train such models? A sensible criterion would seem to be to maximize the probability that we predict the correct label in response to each example. Assuming each example is labeled independently from others, this probability of assigning correct labels to examples is given by the product

$$L(\theta, \theta_0) = \prod_{t=1}^{n} P(y_t|\mathbf{x}_t, \theta, \theta_0) \tag{9}$$

$L(\theta, \theta_0)$ is known as the (conditional) likelihood function and is interpreted as a function of the parameters for a fixed data (labels and examples). By maximizing this conditional likelihood with respect to θ and θ_0 we obtain *maximum likelihood estimates* of the parameters. Maximum likelihood *estimators*[1] have many nice properties. For example, assuming we have selected the right model class (logistic regression model) and certain regularity conditions hold, then the ML estimator is a) *consistent* (we will get the right parameter values in the limit of a large number of training examples), and b) *efficient* (no other estimator will converge to the correct parameter values faster in the mean squared sense). But what if we do not have the right model class? Neither property may hold as a result. More robust estimators can be found in a larger class of estimators called *M-estimators* that includes maximum likelihood. We will nevertheless use the maximum likelihood principle to set the parameter values.

The product form of the conditional likelihood function is a bit difficult to work with directly so we will maximize its logarithm instead:

$$l(\theta, \theta_0) = \sum_{t=1}^{n} \log P(y_t | \mathbf{x}_t, \theta, \theta_0) \tag{10}$$

Alternatively, we can *minimize* the negative logarithm

$$\begin{aligned} -l(\theta, \theta_0) &= \sum_{t=1}^{n} \overbrace{-\log P(y_t | \mathbf{x}_t, \theta, \theta_0)}^{\text{log-loss}} && (11) \\ &= \sum_{t=1}^{n} -\log g\left(y_t(\theta^T \mathbf{x}_t + \theta_0) \right) && (12) \\ &= \sum_{t=1}^{n} \log\left[1 + \exp\left(-y_t(\theta^T \mathbf{x}_t + \theta_0) \right) \right] && (13) \end{aligned}$$

We can interpret this similarly to the sum of the hinge losses in the SVM approach. As before, we have a base loss function, here $\log[1+\exp(-z)]$ (Figure 1b), similar to the hinge loss (Figure 1a), and this loss depends only on the value of the "margin" $y_t(\theta^T \mathbf{x}_t + \theta_0)$ for each example. The difference here is that we have a clear probabilistic interpretation of the "strength" of the prediction, i.e., how high $P(y_t | \mathbf{x}_t, \theta, \theta_0)$ is for any particular example. Having a probabilistic interpretation does not, however, mean that the probability values are in any way sensible or *calibrated*. Predicted probabilities are *calibrated* when they

[1] An *estimator* is a function that maps data to parameter values. An *estimate* is the value obtained in response to specific data.

correspond to observed frequencies. So, for example, if we group together all the examples for which we predict positive label with probability 0.5, then roughly half of them should be labeled +1. Probability estimates are rarely well-calibrated but can nevertheless be useful.

The minimization problem we have defined above is convex and there are a number of optimization methods available for finding the minimizing $\hat{\theta}$ and $\hat{\theta}_0$ including simple gradient descent. In a simple (stochastic) gradient descent, we would modify the parameters in response to each term in the sum (based on each training example). To specify the updates we need the following derivatives

$$\frac{d}{d\theta_0} \log \left[1 + \exp\left(-y_t(\theta^T \mathbf{x}_t + \theta_0) \right)\right] = -y_t \frac{\exp\left(-y_t(\theta^T \mathbf{x}_t + \theta_0)\right)}{1 + \exp\left(-y_t(\theta^T \mathbf{x}_t + \theta_0) \right)} \quad (14)$$

$$= -y_t[1 - P(y_t|\mathbf{x}_t, \theta, \theta_0)] \quad (15)$$

and

$$\frac{d}{d\theta} \log \left[1 + \exp\left(-y_t(\theta^T \mathbf{x}_t + \theta_0) \right)\right] = -y_t \mathbf{x}_t[1 - P(y_t|\mathbf{x}_t, \theta, \theta_0)] \quad (16)$$

The parameters are then updated by selecting training examples at random and moving the parameters in the opposite direction of the derivatives:

$$\theta_0 \leftarrow \theta_0 + \eta \cdot y_t[1 - P(y_t|\mathbf{x}_t, \theta, \theta_0)] \quad (17)$$

$$\theta \leftarrow \theta + \eta \cdot y_t \mathbf{x}_t[1 - P(y_t|\mathbf{x}_t, \theta, \theta_0)] \quad (18)$$

where η is a small (positive) learning rate. Note that $P(y_t|\mathbf{x}_t, \theta, \theta_0)$ is the probability that we predict the training label correctly and $[1 - P(y_t|\mathbf{x}_t, \theta, \theta_0)]$ is the probability of making a mistake. The stochastic gradient descent updates in the logistic regression context therefore strongly resemble the perceptron mistake driven updates. The key difference here is that the updates are graded, made in proportion to the probability of making a mistake.

The stochastic gradient descent algorithm leads to no significant change on average when the gradient of the full objective equals zero. Setting the gradient to zero is also a necessary condition of optimality:

$$\frac{d}{d\theta_0}(-l(\theta, \theta_0) = -\sum_{t=1}^{n} y_t[1 - P(y_t|\mathbf{x}_t, \theta, \theta_0)] = 0 \quad (19)$$

$$\frac{d}{d\theta}(-l(\theta, \theta_0)) = -\sum_{t=1}^{n} y_t \mathbf{x}_t[1 - P(y_t|\mathbf{x}_t, \theta, \theta_0)] = 0 \quad (20)$$

The sum in Eq.(19) is the difference between mistake probabilities associated with positively and negatively labeled examples. The optimality of θ_0 therefore ensures that the mistakes are balanced in this (soft) sense. Another way of understanding this is that the vector of mistake probabilities is orthogonal to the vector of labels. Similarly, the optimal setting of θ is characterized by mistake probabilities that are orthogonal to all rows of the label-example matrix $\tilde{\mathbf{X}} = [y_1\mathbf{x}_1, \ldots, y_n\mathbf{x}_n]$. In other words, for each dimension j of the example vectors, $[y_1 x_{1j}, \ldots, y_n x_{nj}]$ is orthogonal to the mistake probabilities. Taken together, these orthogonality conditions ensure that there's no further linearly available information in the examples to improve the predicted probabilities (or mistake probabilities). This is perhaps a bit easier to see if we first map ± 1 labels into 0/1 labels: $\tilde{y}_t = (1 + y_t)/2$ so that $\tilde{y}_t \in \{0, 1\}$. Then the above optimality conditions can be rewritten in terms of prediction errors $[\tilde{y}_t - P(y = 1|\mathbf{x}_t, \theta, \theta_0)]$ rather than mistake probabilities as

$$\sum_{t=1}^{n} [\tilde{y}_t - P(y = 1|\mathbf{x}_t, \theta, \theta_0)] = 0 \tag{21}$$

$$\sum_{t=1}^{n} \mathbf{x}_t [\tilde{y}_t - P(y = 1|\mathbf{x}_t, \theta, \theta_0)] = 0 \tag{22}$$

and

$$\theta_0' \sum_{t=1}^{n} [\tilde{y}_t - P(y = 1|\mathbf{x}_t, \theta, \theta_0)] + \theta'^T \sum_{t=1}^{n} \mathbf{x}_t [\tilde{y}_t - P(y = 1|\mathbf{x}_t, \theta, \theta_0)] \tag{23}$$

$$= \sum_{t=1}^{n} (\theta'^T \mathbf{x}_t + \theta_0)[\tilde{y}_t - P(y = 1|\mathbf{x}_t, \theta, \theta_0)] = 0 \tag{24}$$

meaning that the prediction errors are orthogonal to any linear function of the inputs.

Let's try to briefly understand the type of predictions we could obtain via maximum likelihood estimation of the logistic regression model. Suppose the training examples are linearly separable. In this case we can find parameter values such that $y_t(\theta^T \mathbf{x}_t + \theta_0)$ are positive for all training examples. By scaling up the parameters, we make these values larger and larger. This is beneficial as far as the likelihood model is concerned since the log of the logistic function is strictly increasing as a function of $y_t(\theta^T \mathbf{x}_t + \theta_0)$ (the loss $\log\left[1 + \exp\left(-y_t(\theta^T \mathbf{x}_t + \theta_0)\right)\right]$ is strictly decreasing). Thus, as a result, the maximum likelihood parameter values would become unbounded, and infinite scaling of any parameters corresponding to a perfect linear classifier would attain the highest likelihood (likelihood of exactly one or the loss exactly zero). The resulting probability values, predicting each training label correctly with probability one, are hardly accurate in the sense of reflecting

our uncertainty about what the labels might be. So, when the number of training examples is small we would need to add the regularizer $\|\theta\|^2/2$ just as in the SVM model. The regularizer helps select reasonable parameters when the available training data fails to sufficiently constrain the linear classifier.

To estimate the parameters of the logistic regression model with regularization we would minimize instead

$$\frac{1}{2}\|\theta\|^2 + C\sum_{t=1}^{n} \log\left[1 + \exp\left(-y_t(\theta^T \mathbf{x}_t + \theta_0)\right)\right] \tag{25}$$

where the constant C again specifies the trade-off between correct classification (the objective) and the regularization penalty. The regularization problem is typically written (equivalently) as

$$\frac{\lambda}{2}\|\theta\|^2 + \sum_{t=1}^{n} \log\left[1 + \exp\left(-y_t(\theta^T \mathbf{x}_t + \theta_0)\right)\right] \tag{26}$$

since it seems more natural to vary the strength of regularization with λ while keeping the objective the same.

Linear regression, active learning

We arrived at the logistic regression model when trying to explicitly model the uncertainty about the labels in a linear classifier. The same general modeling approach permits us to use linear predictions in various other contexts as well. The simplest of them is regression where the goal is to predict a continuous response $y_t \in \mathcal{R}$ to each example vector. Here too focusing on linear predictions won't be inherently limiting as linear predictions can be easily extended (next lecture).

So, how should we model continuous responses? The linear function of the input already produces a "mean prediction" or $\theta^T \mathbf{x} + \theta_0$. By treating this as a mean prediction more formally, we are stating that the expected value of the response variable, conditioned on $\mathbf{x}$, is $\theta^T \mathbf{x} + \theta_0$. More succinctly, we say that $E\{y|\mathbf{x}\} = \theta^T \mathbf{x} + \theta_0$. It remains to associate a distribution over the responses around such mean prediction. The simplest symmetric distribution is the normal (Gaussian) distribution. In other words, we say that the responses y follow the normal pdf

$$N(y;\, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y-\mu)^2 \right) \tag{1}$$

where $\mu = \theta^T \mathbf{x} + \theta_0$. Our response model is therefore defined as

$$P(y|\mathbf{x}, \theta, \theta_0) = N(y;\, \theta^T \mathbf{x} + \theta_0,\, \sigma^2) \tag{2}$$

So, when the input is 1-dimensional, we predict a mean response that is a line in (x, y) space, and assume that noise in y is normally distributed with zero mean and variance σ^2. Note that the noise variance σ^2 in the model does not depend on the input x. Moreover, we only model variation in the y-direction while expecting to know x with perfect precision. Taking into account the effect of potential noise in x on the responses y would tie parameters θ and θ_0 to the noise variance σ^2, potentially in an input dependent manner. The specifics of this coupling depend on the form of noise added to x. We will discuss this in a bit more detail later on.

We can also write the linear regression model in another way to explicate how exactly the additive noise appears in the responses:

$$y = \theta^T \mathbf{x} + \theta_0 + \epsilon \tag{3}$$

where $\epsilon \sim N(0, \sigma^2)$ (meaning that noise ϵ is distributed normally with mean zero and variance σ^2). Clearly for this model $E\{y|\mathbf{x}\} = \theta^T \mathbf{x} + \theta_0$ since ϵ has zero mean. Moreover, adding Gaussian noise to a deterministic prediction $\theta^T \mathbf{x} + \theta_0$ makes y normally distributed

with mean $\theta^T \mathbf{x} + \theta_0$ and variance σ^2, as before. So, in particular, for the training inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and outputs $y_1, \ldots, y_n$, the model relating them is

$$y_t = \theta^T \mathbf{x}_t + \theta_0 + \epsilon_t, \quad t = 1, \ldots, n \tag{4}$$

where $e_t \sim N(0, \sigma^2)$ and e_i is independent of e_j for any $i \neq i$.

Regardless of how we choose the write the model (both forms are useful) we can find the parameter estimates by maximizing the conditional likelihood. Similarly to the logistic regression case, the conditional likelihood is written as

$$L(\theta, \theta_0, \sigma^2) = \prod_{t=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y_t - \theta^T \mathbf{x}_t - \theta_0)^2 \right) \tag{5}$$

Note that σ^2 is also a parameter we have to estimate. It accounts for errors not captured by the linear model. In terms of the log-likelihood, we try to maximize

$$\begin{aligned} l(\theta, \theta_0, \sigma^2) &= \sum_{t=1}^{n} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y_t - \theta^T \mathbf{x}_t - \theta_0)^2 \right) \right] &(6)\\ &= \sum_{t=1}^{n} \left[-\frac{1}{2}\log(2\pi) - \frac{1}{2}\log\sigma^2 - \frac{1}{2\sigma^2}(y_t - \theta^T \mathbf{x}_t - \theta_0)^2 \right] &(7)\\ &= \text{const.} - \frac{n}{2}\log\sigma^2 - \frac{1}{2\sigma^2}\sum_{t=1}^{n}(y_t - \theta^T \mathbf{x}_t - \theta_0)^2 &(8) \end{aligned}$$

where 'const.' absorbs terms that do not depend on the parameters. Now, the problem of estimating the parameters θ and θ_0 is nicely decoupled from estimating σ^2. In other words, we can find the *maximizing* $\hat{\theta}$ and $\hat{\theta}_0$ by simply *minimizing* the mean squared error

$$\sum_{t=1}^{n}(y_t - \theta^T \mathbf{x}_t - \theta_0)^2 \tag{9}$$

It is perhaps easiest to write the solution based on a bit of matrix calculation. Let $\mathbf{X}$ be a matrix whose rows, indexed by training examples, are given by $[\mathbf{x}_t^T, 1]$ ($\mathbf{x}_t$ turned into a row vector and 1 added at the end). In terms of this matrix, the minimization problem

becomes

$$\sum_{t=1}^{n}\left(y_t - \left[\begin{array}{c}\theta\\ \theta_0\end{array}\right]^T \left[\begin{array}{c}\mathbf{x}_t\\ 1\end{array}\right]\right)^2 = \sum_{t=1}^{n}\left(y_t - \left[\mathbf{x}_t^T, 1\right]\left[\begin{array}{c}\theta\\ \theta_0\end{array}\right]\right)^2 \quad (10)$$

$$= \left\|\left[\begin{array}{c}y_1\\ \cdots\\ y_n\end{array}\right] - \left[\begin{array}{c}\mathbf{x}_1^T, 1\\ \cdots\\ \mathbf{x}_n^T, 1\end{array}\right]\left[\begin{array}{c}\theta\\ \theta_0\end{array}\right]\right\|^2 \quad (11)$$

$$= \left\|\mathbf{y} - \mathbf{X}\left[\begin{array}{c}\theta\\ \theta_0\end{array}\right]\right\|^2 \quad (12)$$

$$= \mathbf{y}^T\mathbf{y} - 2\left[\begin{array}{c}\theta\\ \theta_0\end{array}\right]^T \mathbf{X}^T\mathbf{y} + \left[\begin{array}{c}\theta\\ \theta_0\end{array}\right]^T \mathbf{X}^T\mathbf{X}\left[\begin{array}{c}\theta\\ \theta_0\end{array}\right] \quad (13)$$

where $\mathbf{y} = [y_1, \ldots, y_n]^T$ is a vector of training responses. Solving it yields

$$\left[\begin{array}{c}\hat{\theta}\\ \hat{\theta}_0\end{array}\right] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \quad (14)$$

Note that the optimal parameter values are linear functions of the observed responses $\mathbf{y}$. We will make use of this property later on. The dependence on the training inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$ (or the matrix $\mathbf{X}$) is non-linear, however.

The noise variance can be subsequently set to account for the remaining prediction errors. Indeed, the the maximizing value of σ^2 is given by

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{\theta}^T\mathbf{x}_t - \hat{\theta}_0)^2 \quad (15)$$

which is the average squared prediction error. Note that we cannot compute $\hat{\sigma}^2$ before knowing how well the linear model explains the responses.

Bias and variance of the parameter estimates

We can make use of the closed form parameter estimates in Eq.(14) to analyze how good these estimates are. For this purpose let's make the strong assumption that the actual relation between $\mathbf{x}$ and y follows a linear model of the same type that we are estimating (we just don't know the correct parameter values θ^*, θ_0^*, and σ^{*2}). We can therefore describe the observed responses y_t as

$$y_t = \theta^{*T}\mathbf{x}_t + \theta_0^* + \epsilon_t, \quad t = 1, \ldots, n \quad (16)$$

where $\epsilon_t \sim N(0, \sigma^{*2})$. In a matrix form

$$\mathbf{y} = \mathbf{X} \left[\begin{array}{c} \theta^* \\ \theta_0^* \end{array} \right] + \mathbf{e} \tag{17}$$

where $\mathbf{e} = [\epsilon_1, \ldots, \epsilon_n]^T$, $E\{\mathbf{e}\} = 0$ and $E\{\mathbf{e}\mathbf{e}^T\} = \sigma^{*2}\,\mathbf{I}$. The noise vector $\mathbf{e}$ is also independent of the inputs or $\mathbf{X}$. Plugging this form of responses into Eq.(14) we get

$$\begin{aligned}
\left[\begin{array}{c} \hat{\theta} \\ \hat{\theta}_0 \end{array} \right] &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{X} \left[\begin{array}{c} \theta^* \\ \theta_0^* \end{array} \right] + \mathbf{e}) && (18)\\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X} \left[\begin{array}{c} \theta^* \\ \theta_0^* \end{array} \right] + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{e} && (19)\\
&= \left[\begin{array}{c} \theta^* \\ \theta_0^* \end{array} \right] + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{e} && (20)
\end{aligned}$$

In other words, our parameter estimates can be decomposed into the sum of correct underlying parameters and estimates based on noise alone (i.e., based on $\mathbf{e}$). Thus, on average with fixed inputs

$$E\{\left[\begin{array}{c} \hat{\theta} \\ \hat{\theta}_0 \end{array} \right] |\mathbf{X}\} = \left[\begin{array}{c} \theta^* \\ \theta_0^* \end{array} \right] + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T E\{\mathbf{e}|\mathbf{X}\} = \left[\begin{array}{c} \theta^* \\ \theta_0^* \end{array} \right] \tag{21}$$

Our parameter estimates are therefore *unbiased* or correct on average when averaging is over possible training sets we could generate. The averaging here is conditioned on the specific training inputs.

Using Eq.(20) and Eq.(21) we can also evaluate the conditional co-variance of the parameter estimates where the expectation is again over the noise in the outputs:

$$\begin{aligned}
Cov\{\left[\begin{array}{c} \hat{\theta} \\ \hat{\theta}_0 \end{array} \right] |\mathbf{X}\} &= E\left\{ \left(\left[\begin{array}{c} \hat{\theta} \\ \hat{\theta}_0 \end{array} \right] - \left[\begin{array}{c} \theta^* \\ \theta_0^* \end{array} \right] \right) \left(\left[\begin{array}{c} \hat{\theta} \\ \hat{\theta}_0 \end{array} \right] - \left[\begin{array}{c} \theta^* \\ \theta_0^* \end{array} \right] \right)^T |\mathbf{X} \right\} && (22)\\
&= E\left\{ \left((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{e}\right) \left((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{e}\right)^T |\mathbf{X} \right\} && (23)\\
&= E\left\{ (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{e}\mathbf{e}^T\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} | \mathbf{X} \right\} && (24)\\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\, E\{\mathbf{e}\mathbf{e}^T|\mathbf{X}\}\, \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} && (25)\\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\, (\sigma^{*2}\,\mathbf{I})\, \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} && (26)\\
&= \sigma^{*2}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} && (27)\\
&= \sigma^{*2}(\mathbf{X}^T\mathbf{X})^{-1} && (28)
\end{aligned}$$

So the way in which the parameters vary in response to noise in the outputs is a function of the inputs or $\mathbf{X}$. We will use this property in the next section to select inputs so as to improve the quality of the parameter estimates or to reduce the variance of predictions.

Based on the bias and variance calculations we can evaluate the mean squared error of the parameter estimates. To this end, we use the fact that the expectation of the squared norm of any vector valued random variable can be decomposed into a bias and variance components as follows:

$$
\begin{aligned}
E\left\{ \|\mathbf{z} - \mathbf{z}^*\|^2 \right\} &= E\left\{ \|\mathbf{z} - E\{\mathbf{z}\} + E\{\mathbf{z}\} - \mathbf{z}^*\|^2 \right\} && (29)\\
&= E\left\{ \|\mathbf{z} - E\{\mathbf{z}\}\|^2 + 2(\mathbf{z} - E\{\mathbf{z}\})^T (E\{\mathbf{z}\} - \mathbf{z}^*) + \|E\{\mathbf{z}\} - \mathbf{z}^*\|^2 \right\} && (30)\\
&= E\left\{ \|\mathbf{z} - E\{\mathbf{z}\}\|^2 \right\} + 2E\left\{ (\mathbf{z} - E\{\mathbf{z}\})^T \right\} (E\{\mathbf{z}\} - \mathbf{z}^*) + \|E\{\mathbf{z}\} - \mathbf{z}^*\|^2 \\
&= \overbrace{E\left\{ \|\mathbf{z} - E\{\mathbf{z}\}\|^2 \right\}}^{\text{variance}} + \overbrace{\|E\{\mathbf{z}\} - \mathbf{z}^*\|^2}^{\text{bias}^2} && (31)
\end{aligned}
$$

where we have assumed that $\mathbf{z}^*$ is fixed. Make sure you understand how this decomposition is derived. We will further elaborate the variance part to better use the result in our context:

$$
\begin{aligned}
E\left\{ \|\mathbf{z} - E\{\mathbf{z}\}\|^2 \right\} &= E\left\{ (\mathbf{z} - E\{\mathbf{z}\})^T (\mathbf{z} - E\{\mathbf{z}\}) \right\} && (32)\\
&= E\left\{ Tr\left[(\mathbf{z} - E\{\mathbf{z}\})^T (\mathbf{z} - E\{\mathbf{z}\}) \right] \right\} && (33)\\
&= E\left\{ Tr\left[(\mathbf{z} - E\{\mathbf{z}\})(\mathbf{z} - E\{\mathbf{z}\})^T \right] \right\} && (34)\\
&= Tr\left[E\left\{ (\mathbf{z} - E\{\mathbf{z}\})(\mathbf{z} - E\{\mathbf{z}\})^T \right\} \right] && (35)\\
&= Tr\left[Cov\{\mathbf{z}\} \right] && (36)
\end{aligned}
$$

where $Tr[\cdot]$ is the matrix *trace*, the sum of its diagonal components, and therefore a linear operation (exchangeable with the expectation). We have also used the fact that $Tr[\mathbf{a}^T\mathbf{b}] = Tr[\mathbf{a}\mathbf{b}^T]$ for any vectors $\mathbf{a}$ and $\mathbf{b}$.

Now, adapting the result to our setting, we get

$$E\left\{ \left\| \begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} - \begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix} \right\|^2 |\mathbf{X} \right\} = \overbrace{Tr\left[Cov\left\{ \begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} |\mathbf{X} \right\} \right]}^{\text{variance}} + \overbrace{\left\| E\left\{ \begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} |\mathbf{X} \right\} - \begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix} \right\|^2}^{\text{bias}^2=0}$$

$$= \sigma^{*2} Tr\left[(\mathbf{X}^T\mathbf{X})^{-1} \right] \quad (37)$$

Let's understand this result a bit further. How does it depend on n, the number of training examples? In other words, how quickly does the mean squared error decrease as the number of training examples increases, assuming the input examples $\mathbf{x}$ are sampled independently from some underlying distribution $P(\mathbf{x})$? To answer this let's start by analyzing what happens to the matrix $\mathbf{X}^T\mathbf{X}$:

$$\mathbf{X}^T\mathbf{X} = \sum_{t=1}^{n} \begin{bmatrix} \mathbf{x}_t \\ 1 \end{bmatrix} [\mathbf{x}_t^T, 1] \quad (38)$$

$$= n \cdot \frac{1}{n} \sum_{t=1}^{n} \begin{bmatrix} \mathbf{x}_t \\ 1 \end{bmatrix} [\mathbf{x}_t^T, 1] \quad (39)$$

$$\approx n \cdot E_{\mathbf{x}\sim P}\left\{ \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} [\mathbf{x}^T, 1] \right\} = n \cdot \mathbf{C} \quad (40)$$

where for large n the average will be close to the corresponding expected value. For large n the mean squared error of the parameter estimates will therefore be close to

$$\frac{\sigma^{*2}}{n} \cdot Tr[\mathbf{C}^{-1}] \quad (41)$$

The variance of simply averaging the (noise in the) outputs would behave as σ^{*2}/n. Since we are estimating $d+1$ parameters where d is the input dimension, this dependence would have to be in $Tr[\mathbf{C}^{-1}]$. Indeed it is. This term, a trace of a $(d+1) \times (d+1)$ matrix $\mathbf{C}^{-1}$, is directly proportional to $d+1$.

Penalized log-likelihood and Ridge regression

When the number of training examples is small, i.e., not too much larger than the number of parameters (dimension of the inputs), it is often beneficial to *regularize* the parameter estimates. We will derive the form of regularization here by assigning a prior distribution over the parameters $P(\theta, \theta_0)$. The purpose of the prior is to prefer small parameter values

(predict values close to zero) in the absence of data. Specifically, we will look at simple zero mean Gaussian distributions

$$P(\theta, \theta_0; \sigma'^2) = N(\begin{bmatrix} \theta \\ \theta_0 \end{bmatrix}; \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \sigma'^2 \mathbf{I}) = N(\theta_0; 0, \sigma'^2) \prod_{j=1}^{d} N(\theta_j; 0, \sigma'^2) \tag{42}$$

where the variance parameter σ'^2 in the prior distribution specifies how strongly we wish to bias the parameters towards zero.

By combining the log-likelihood criterion with the prior we obtain a *penalized log-likelihood* function (penalized by the prior):

$$\begin{aligned} l'(\theta, \theta_0, \sigma^2) &= \sum_{t=1}^{n} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y_t - \theta^T \mathbf{x}_t - \theta_0)^2 \right) \right] + \log P(\theta, \theta_0; \sigma'^2) \quad (43) \\ &= \text{const.} - \frac{n}{2} \log \sigma^2 - \frac{1}{2\sigma^2} \sum_{t=1}^{n} (y_t - \theta^T \mathbf{x}_t - \theta_0)^2 \\ &\quad - \frac{1}{2\sigma'^2}(\theta_0^2 + \sum_{j=1}^{d} \theta_j^2) - \frac{d+1}{2} \log \sigma'^2 \quad (44) \end{aligned}$$

It is convenient to tie the prior variance σ'^2 to the noise variance σ^2 according to $\sigma'^2 = \sigma^2/\lambda$. This has the effect that if the noise variance σ^2 is large, we penalize the parameters very little (permit large deviations from zero by assuming a large σ'^2). On the other hand, if the noise variance is small, we could be *over-fitting* the linear model. This happens, for example, when the number of training examples is small. In this case most of the responses can be explained directly by the linear model making the noise variance very small. In such cases our penalty for the parameters will be larger as well (prior variance is smaller).

Incorporating this parameter tie into the penalized log-likelihood function gives

$$\begin{aligned} l'(\theta, \theta_0, \sigma^2) &= \text{const.} - \frac{n}{2} \log \sigma^2 - \frac{1}{2\sigma^2} \sum_{t=1}^{n} (y_t - \theta^T \mathbf{x}_t - \theta_0)^2 \\ &\quad - \frac{\lambda}{2\sigma^2}(\theta_0^2 + \sum_{j=1}^{d} \theta_j^2) - \frac{d+1}{2} \log(\sigma^2/\lambda) \quad (45) \\ &= \text{const.} - \frac{n+d+1}{2} \log \sigma^2 + \frac{d+1}{2} \log \lambda \quad (46) \\ &\quad - \frac{1}{2\sigma^2} \left[\sum_{t=1}^{n} (y_t - \theta^T \mathbf{x}_t - \theta_0)^2 + \lambda(\theta_0^2 + \sum_{j=1}^{d} \theta_j^2) \right] \quad (47) \end{aligned}$$

where again the estimation of θ and θ_0 separates from setting the noise variance σ^2. Note that this separation is achieved because we tied the prior and noise variance parameters. The above regularized problem of finding the parameter estimates $\hat{\theta}$ and $\hat{\theta}_0$ is known as *Ridge regression*.

As before, we can get closed form estimates for the parameters (we omit the analogous derivation):

$$\left[\begin{array}{c} \hat{\theta} \\ \hat{\theta}_0 \end{array} \right] = (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \tag{48}$$

It is now useful to understand how the properties of these parameter estimates depend on λ. For example, are the parameter estimates *unbiased*? No, they are not:

$$\begin{aligned} E\left\{ \left[\begin{array}{c} \hat{\theta} \\ \hat{\theta}_0 \end{array} \right] | \mathbf{X} \right\} &= (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{X} \left[\begin{array}{c} \theta^* \\ \theta_0^* \end{array} \right] & (49) \\ &= (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I} - \lambda \mathbf{I}) \left[\begin{array}{c} \theta^* \\ \theta_0^* \end{array} \right] & (50) \\ &= \left[\begin{array}{c} \theta^* \\ \hat{\theta}_0^* \end{array} \right] \overbrace{-\lambda (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} \left[\begin{array}{c} \theta^* \\ \theta_0^* \end{array} \right]}^{\text{bias}} & (51) \\ &= \left(\mathbf{I} - \lambda (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} \right) \left[\begin{array}{c} \theta^* \\ \theta_0^* \end{array} \right] & (52) \end{aligned}$$

It is straightforward to check that $\left(\mathbf{I} - \lambda (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} \right)$ is a positive definite matrix with eigenvalues all less than one. The parameter estimates are therefore shrunk towards zero and more so the larger the value of λ. This is what we would expect since we explicitly favored small parameter values with the prior penalty. What do we gain from such *biased* parameter estimates? Let's evaluate the mean squared error, starting with the covariance:

$$\begin{aligned} Cov\left\{ \left[\begin{array}{c} \hat{\theta} \\ \hat{\theta}_0 \end{array} \right] | \mathbf{X} \right\} &= \sigma^{*2} (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{X} (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} & (53) \\ &= \sigma^{*2} (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X} - \lambda \mathbf{I}) (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} & (54) \\ &= \sigma^{*2} (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} - \lambda \sigma^{*2} (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-2} & (55) \end{aligned}$$

The mean squared error in the parameters is therefore given by (we again omit the deriva-

tion that can be obtained similarly to previous expressions):

$$E\left\{\left\|\begin{bmatrix}\hat{\theta}\\ \hat{\theta}_0\end{bmatrix}-\begin{bmatrix}\theta^*\\ \theta_0^*\end{bmatrix}\right\|^2|\mathbf{X}\right\} = \sigma^{*2}\cdot Tr\left[(\lambda\mathbf{I}+\mathbf{X}^T\mathbf{X})^{-1}-\lambda(\lambda\mathbf{I}+\mathbf{X}^T\mathbf{X})^{-2}\right] + \lambda^2\begin{bmatrix}\theta^*\\ \theta_0^*\end{bmatrix}^T(\lambda\mathbf{I}+\mathbf{X}^T\mathbf{X})^{-2}\begin{bmatrix}\theta^*\\ \theta_0^*\end{bmatrix} \quad (56)$$

Can this be smaller than the mean squared error corresponding to the unregularized estimates $\sigma^{*2}\cdot Tr\left[(\mathbf{X}^T\mathbf{X})^{-1}\right]$? Yes, it can. This is indeed the benefit from regularization: we can reduce large variance at the cost of introducing a bit of bias. We will get back to this trade-off in the context of *model selection*.

Let's exemplify the effect of λ on the mean squared error in a context of a very simple 1-dimensional example. Suppose, we have observed responses for only two points, $x=-1$ and $x=1$. In this case,

$$\mathbf{X}=\begin{bmatrix}-1 & 1\\ 1 & 1\end{bmatrix},\ \mathbf{X}^T\mathbf{X}=\begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix},\ (\lambda\mathbf{I}+\mathbf{X}^T\mathbf{X})^{-1}=\begin{bmatrix}1/(2+\lambda) & 0\\ 0 & 1/(2+\lambda)\end{bmatrix} \quad (57)$$

The expression for the mean squared error therefore becomes

$$\begin{aligned}E\left\{\left\|\begin{bmatrix}\hat{\theta}\\ \hat{\theta}_0\end{bmatrix}-\begin{bmatrix}\theta^*\\ \theta_0^*\end{bmatrix}\right\|^2|\mathbf{X}\right\} &= \sigma^{*2}\left(\frac{2}{(2+\lambda)}-\frac{2\lambda}{(2+\lambda)^2}\right)+\frac{\lambda^2}{(2+\lambda)^2}(\theta^{*2}+\theta_0^{*2})\\ &= \frac{4\sigma^{*2}}{(2+\lambda)^2}+\frac{\lambda^2}{(2+\lambda)^2}(\theta^{*2}+\theta_0^{*2}) \quad (58)\end{aligned}$$

We should compare this to $\sigma^{*2}Tr\left[(\mathbf{X}^T\mathbf{X})^{-1}\right]=\sigma^{*2}$ obtained without regularization (corresponds to setting $\lambda=0$). In the noisy case $\sigma^{*2}>\theta^{*2}+\theta_0^{*2}$ we can set $\lambda=2$ and obtain

$$E\left\{\left\|\begin{bmatrix}\hat{\theta}\\ \hat{\theta}_0\end{bmatrix}-\begin{bmatrix}\theta^*\\ \theta_0^*\end{bmatrix}\right\|^2|\mathbf{X}\right\}=\frac{4\sigma^{*2}}{16}+\frac{4}{16}(\theta^{*2}+\theta_0^{*2})<\frac{8\sigma^{*2}}{16}=\frac{1}{2}\sigma^{*2} \quad (59)$$

The mean squared error of the parameters is therefore clearly smaller than without regularization.

Active learning

We can use the expressions for the mean squared error to actively select input points $\mathbf{x}_1,\ldots,\mathbf{x}_n$, when possible, so as to reduce the resulting estimation error. This is an *active*

learning (*experiment design*) problem. The goal is to get by with as few responses as possible without sacrificing the estimation accuracy. For example, in training a classifier we would like to minimize the number of training images we would have to label in order obtain a certain classification accuracy. In the regression context, we may be selecting among possible experiments to carry out (e.g., choosing different operating points for a factory or gauging market/customer responses to different strategies available to us). By letting the method guide the selection of the training examples (inputs), we will generally need far fewer examples in comparison to selecting them at random from some underlying distribution, database, or trying available experiments at random. Fewer responses means less human effort, less cost, or both.

To develop this further let's go back to the unregularized case where

$$E\left\{ \left\| \begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} - \begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix} \right\|^2 |\mathbf{X} \right\} = \sigma^{*2} Tr\left[(\mathbf{X}^T\mathbf{X})^{-1} \right] \tag{60}$$

We do not know the noise variance σ^{*2} for the correct model but it only appears as a multiplicative constant in the above expression and therefore won't affect how we should choose the inputs. When the choice of inputs is indeed up to us (e.g., which experiments to carry out) we can select them so as to minimize $Tr\left[(\mathbf{X}^T\mathbf{X})^{-1} \right]$. One caveat of this approach is that it relies on the underlying relationship between the inputs and the responses to be linear. When this is no longer the case we may end up with clearly suboptimal selections.

We will discuss next time how we can find say n input examples $\mathbf{x}_1, \ldots, \mathbf{x}_n$ that minimize the criterion.

Lecture topics:

- Active learning
- Non-linear predictions, kernels

Active learning

We can use the expressions for the mean squared error to actively select input points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, when possible, so as to reduce the resulting estimation error. This is an *active learning* (*experiment design*) problem. By letting the method guide the selection of the training examples (inputs), we will generally need far fewer examples in comparison to selecting them at random from some underlying distribution, database, or trying available experiments at random.

To develop this further, recall that we continue to assume that the responses y come from some linear model $y = \theta^{*T}\mathbf{x} + \theta_0^* + \epsilon$ where $\epsilon \sim N(0, \sigma^{*2})$. Nothing is assumed about the distribution of $\mathbf{x}$ as the choice of the inputs is in our control. For any given set of inputs, $\mathbf{x}_1, \ldots, \mathbf{x}_n$, we derived last time an expression for the mean squared error of the maximum likelihood parameter estimates $\hat{\theta}$ and $\hat{\theta}_0$:

$$E\left\{ \left\| \begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} - \begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix} \right\|^2 | \mathbf{X} \right\} = \sigma^{*2} Tr\left[(\mathbf{X}^T\mathbf{X})^{-1} \right] \tag{1}$$

where the expectation is relative to the responses generated from the underlying linear model, i.e., over different training sets generated from the linear model. We do not know the noise variance σ^{*2} for the correct model but it only appears as a multiplicative constant in the above expression and therefore won't affect how we should choose the inputs. When the choice of inputs is indeed up to us (e.g., which experiments to carry out) we can select them so as to minimize $Tr\left[(\mathbf{X}^T\mathbf{X})^{-1} \right]$. One caveat of this approach is that it relies on the underlying relationship between the inputs and the responses to be linear. When this is no longer the case we may end up with clearly suboptimal selections.

Given the selection criterion, how should we find say n input examples $\mathbf{x}_1, \ldots, \mathbf{x}_n$ that minimize it? One simple approach is to select them one after the other, merely optimizing the selection of the next one in light of what we already have. Let's assume then that we already have $\mathbf{X}$ and thus have $\mathbf{A} = (\mathbf{X}^T\mathbf{X})^{-1}$ (assuming it is already invertible). We are trying to select another input example $\mathbf{x}$ that adds a row $[\mathbf{x}^T, 1]$ to $\mathbf{X}$. In an applied context we are typically constrained by what $\mathbf{x}$ can be (e.g., due to the experimental setup). We

will discuss simple constraints below. Let's now evaluate the effect of adding a new (valid) row:

$$\left[\begin{array}{c} \mathbf{X} \\ \mathbf{x}^T\ 1 \end{array}\right]^T \left[\begin{array}{c} \mathbf{X} \\ \mathbf{x}^T\ 1 \end{array}\right] = (\mathbf{X}^T\mathbf{X}) + \left[\begin{array}{c} \mathbf{x} \\ 1 \end{array}\right]\left[\begin{array}{c} \mathbf{x} \\ 1 \end{array}\right]^T = \mathbf{A}^{-1} + \mathbf{v}\mathbf{v}^T \tag{2}$$

where $\mathbf{v} = [\mathbf{x}^T, 1]^T$. We would like to find a valid $\mathbf{v}$ that minimizes

$$Tr\left[(\mathbf{A}^{-1} + \mathbf{v}\mathbf{v}^T)^{-1}\right] \tag{3}$$

The matrix inverse can actually be carried out in closed form (easy enough to check)

$$(\mathbf{A}^{-1} + \mathbf{v}\mathbf{v}^T)^{-1} = \mathbf{A} - \frac{1}{(1 + \mathbf{v}^T\mathbf{A}\mathbf{v})}\mathbf{A}\mathbf{v}\mathbf{v}^T\mathbf{A} \tag{4}$$

so that the trace becomes

$$\begin{aligned} Tr\left[(\mathbf{A}^{-1} + \mathbf{v}\mathbf{v}^T)^{-1}\right] &= Tr\left[\mathbf{A}\right] - \frac{1}{(1 + \mathbf{v}^T\mathbf{A}\mathbf{v})}Tr\left[\mathbf{A}\mathbf{v}\mathbf{v}^T\mathbf{A}\right] & (5)\\ &= Tr\left[\mathbf{A}\right] - \frac{1}{(1 + \mathbf{v}^T\mathbf{A}\mathbf{v})}Tr\left[\mathbf{v}^T\mathbf{A}\mathbf{A}\mathbf{v}\right] & (6)\\ &= Tr\left[\mathbf{A}\right] - \frac{\mathbf{v}^T\mathbf{A}\mathbf{A}\mathbf{v}}{(1 + \mathbf{v}^T\mathbf{A}\mathbf{v})} & (7) \end{aligned}$$

Note that since $Tr[\mathbf{A}] = Tr[(\mathbf{X}^T\mathbf{X})^{-1}]$ is the mean squared error before adding the new example, any choice of $\mathbf{v}$, i.e., any additional example $\mathbf{x}$ will reduce the mean squared error. We are interested in finding the one that reduces the error the most. This is the example that maximizes

$$\frac{\mathbf{v}^T\mathbf{A}\mathbf{A}\mathbf{v}}{(1 + \mathbf{v}^T\mathbf{A}\mathbf{v})} \tag{8}$$

How much can we possibly reduce the squared error? The above term is bounded by the largest eigenvalue of $\mathbf{A}$. In other words, with each new example we can at most remove one degree of freedom from the parameter space. If we assume no constraints on the choice of $\mathbf{v}$, the maximizing vector would be of infinite length and proportional to the eigenvector of $\mathbf{A}$ with the largest eigenvalue (all the eigenvalues of $\mathbf{A}$ are positive as it is an inverse of a positive definite matrix $\mathbf{X}^T\mathbf{X}$). It is indeed advantageous in linear regression to have the input points as far from each other as possible (see Figure 1). If we constrain $\|\mathbf{v}\| \leq c$, then the maximizing $\mathbf{v}$ is the normalized eigenvector of $\mathbf{A}$ with the largest eigenvalue,

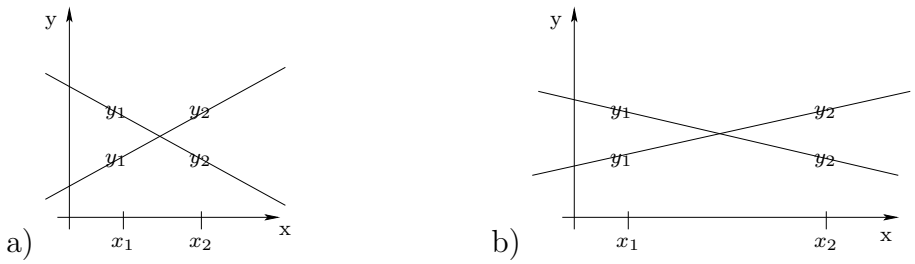

Figure 1: a) The effect of noise in the responses has a large effect on the parameters of the linear model when the corresponding inputs are close to each other; b) the effect is smaller when the inputs are further away.

normalized such that $\|\mathbf{v}\| = c$. Note that it may not be possible to select this eigenvector since $\mathbf{v} = [\mathbf{x}^T, 1]^T$. Other constraints on $\mathbf{x}$ will further restrict $\mathbf{v}$.

Let's take a simple example to illustrate the criterion. Suppose we have a 1-dimensional regression model $y = \theta x + \theta_0 + \epsilon$ where x is constrained to lie within $[-1, 1]$. Assume we have already observed responses for $x_1 = 1$ and $x_2 = -1$. Thus

$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}, \; \mathbf{X}^T\mathbf{X} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \; \mathbf{A} = (\mathbf{X}^T\mathbf{X})^{-1} = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{9}$$

$\mathbf{v} = [x, 1]^T$ and therefore $\mathbf{v}^T\mathbf{A}\mathbf{v} = (x^2 + 1)/2$ and $\mathbf{v}^T\mathbf{A}\mathbf{A}\mathbf{v} = (x^2 + 1)/4$. The criterion to be maximized becomes

$$\frac{\mathbf{v}^T\mathbf{A}\mathbf{A}\mathbf{v}}{(1 + \mathbf{v}^T\mathbf{A}\mathbf{v})} = \frac{(x^2 + 1)/4}{1 + (x^2 + 1)/2} \tag{10}$$

Since $z/(1 + z)$ is an increasing function of z, it follows that the criterion is maximized when $(x^2 + 1)/2$ is maximized. Given the constraints, the maximizing point is $x = 1$ or $x = -1$. Either choice would do but, after selecting one, the other one would be preferred at the next step. The result is consistent with the intuition that for linear models the inputs should be as far from each other as possible (cf. Figure 1).

We have so far used the mean squared error in the parameters as the selection criterion. What about the variance in the predictions? Let's try to find the point $\mathbf{x}$ whose response

we are the most uncertain about. We again write $\mathbf{v} = [\mathbf{x}^T, 1]^T$ so that

$$
\begin{aligned}
Var\{y|\mathbf{x}, \mathbf{X}\} &= E\left\{ \left(\hat{\theta}^T \mathbf{x} + \hat{\theta}_0 - \theta^{*T}\mathbf{x} - {\theta^*}_0 \right)^2 |\mathbf{x}, \mathbf{X} \right\} && (11)\\
&= E\left\{ \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}^T \left(\begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} - \begin{bmatrix} \theta^* \\ \theta^*_0 \end{bmatrix} \right) \left(\begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} - \begin{bmatrix} \theta^* \\ \theta^*_0 \end{bmatrix} \right)^T \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} |\mathbf{x}, \mathbf{X} \right\} && (12)\\
&= \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}^T \sigma^{*2} (\mathbf{X}^T\mathbf{X})^{-1} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} && (13)\\
&= \sigma^{*2} \cdot \mathbf{v}^T \mathbf{A} \mathbf{v} && (14)
\end{aligned}
$$

where the expectation is over responses for existing training examples, again assuming that there is a correct underlying linear model. So the largest variance corresponds to the input $\mathbf{x}$ that maximizes $\mathbf{v}^T\mathbf{A}\mathbf{v}$ where $\mathbf{v} = [\mathbf{x}^T, 1]^T$. In the unconstrained case where there are few or no restrictions on $\mathbf{v}$, this maximizing point is exactly the one we would query according the previous selection criterion.

Non-linear predictions, kernels

Essentially all of what we have discussed can be extended to non-linear models, models that remain linear in the parameters as before but perform non-linear operations in the original input space. This is achieved by mapping the input examples to a higher dimensional feature space where the dimensions in the new feature vectors include non-linear functions of the inputs. The simplest setting for demonstrating this is linear regression in one dimension. Consider therefore the linear model $y = \theta x + \theta_0 + \epsilon$ where $\epsilon \sim N(0, \sigma^2)$. We can obtain a quadratic model by simply mapping the input x to a longer feature vector that includes a term quadratic in x. A third order model can be constructed by including all terms up to degree three, and so on. Explicitly, we would make linear predictions using feature vectors

$$
\begin{aligned}
x &\xrightarrow{\phi} [1, \sqrt{2}x, x^2]^T = \phi(x) && (15)\\
x &\xrightarrow{\phi} [1, \sqrt{3}x, \sqrt{3}x^2, x^3]^T && (16)\\
&\cdots && (17)
\end{aligned}
$$

The role of $\sqrt{2}$ and other constants will become clear shortly. The new polynomial regression model is then given by

$$y = \theta^T \phi(x) + \theta_0 + \epsilon, \quad \epsilon \sim N(0, \sigma^2) \tag{18}$$

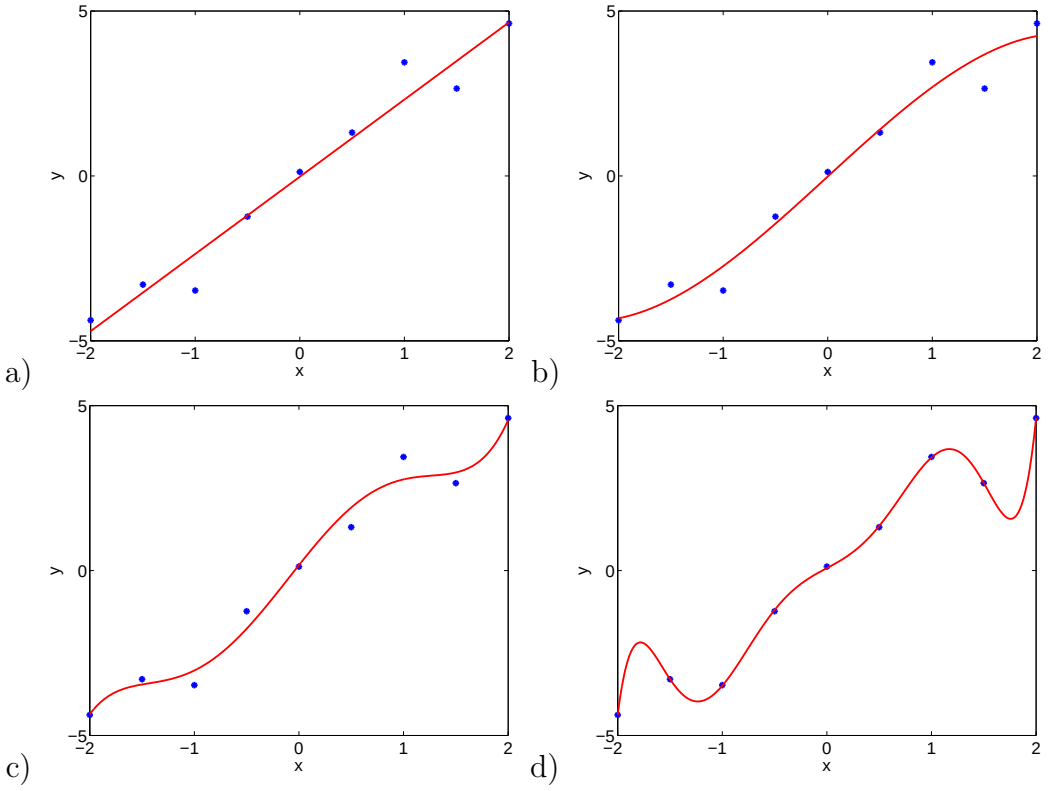

Figure 2: a) a linear model fit; b) a third order polynomial model fit to the same data; c) a fifth order polynomial model; d) a seventh order polynomial model.

where the dimensionality of $\phi(x)$ (and therefore also θ) depends on the order of the polynomial expansion. Regularization of the parameters is almost always used in conjunction with higher dimensional feature vectors. This is necessary since otherwise the higher order models could seriously *overfit* to the data. Examples of fitted polynomial regression models without regularization are shown in Figure 2a-d (the data were generated from a linear model). Note that all these models are linear in the parameters but non-linear in x, save the standard linear regression model in Figure 2a.

The polynomial expansion of input vectors works the same in higher dimensions, e.g.,

$$\mathbf{x} = [x_1, x_2]^T \stackrel{\phi}{\rightarrow} [1, x_1, x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2]^T = \phi(\mathbf{x}) \tag{19}$$

One limitation of explicating the feature vectors is that the dimensionality can increase rapidly with the degree of polynomial expansion, especially when the dimension of the

input vector is already high. The table below gives some indicating of this though the effect is more dramatic with higher input dimensions. Can we somehow avoid explicating the dimensions of the feature vectors? In the context of models we have discussed thus far, yes, we can. We can define the feature vectors implicitly by focusing on specifying the values of their inner products or *kernel* instead. To get a sense of the power of this approach, let's evaluate the inner product between two feature vectors corresponding to the specific cubic expansions of 1-dimensional inputs shown before:

$$\phi(x) = [1, \sqrt{3}x, \sqrt{3}x^2, x^3]^T, \quad (20)$$

$$\phi(x') = [1, \sqrt{3}x', \sqrt{3}x'^2, x'^3]^T, \quad (21)$$

$$\phi(x)^T\phi(x') = 1 + 3xx' + 3(xx')^2 + (xx')^3 = (1 + xx')^3 \quad (22)$$

So it seems we can compactly evaluate the inner products between polynomial feature vectors. The effect is more striking with higher dimensional inputs and higher polynomial degrees. (We did have to specify the constants appropriately in the feature vectors to make this work). To shift the modeling from explicit feature vectors to inner products (kernels) we obviously have to first turn the estimation problem into a form that involves only inner products between feature vectors.

$dim(\mathbf{x}) = 2$

degree p	# of features
2	6
3	10
4	15
5	21

$dim(\mathbf{x}) = 3$

degree p	# of features
2	10
3	20
4	35
5	56

Linear regression and kernels

Let's simplify the model slightly by omitting the offset parameter θ_0, reducing the model to $y = \theta^T\phi(\mathbf{x}) + \epsilon$ where $\phi(\mathbf{x})$ is a particular feature expansion (e.g., polynomial). Our goal here is to turn both the estimation problem and the subsequent prediction task into forms that involve only inner products between the feature vectors.

We have already emphasized that regularization is necessary in conjunction with mapping examples to higher dimensional feature vectors. The regularized least squares objective to be minimized, with parameter λ, is given by

$$J(\theta) = \sum_{t=1}^{n} \left(y_t - \theta^T\phi(\mathbf{x}_t)\right)^2 + \lambda\|\theta\|^2 \quad (23)$$

This form can be derived from penalized log-likelihood estimation (see previous lecture notes). The effect of the regularization penalty is to pull all the parameters towards zero. So any linear dimensions in the parameters that the training feature vectors do not pertain to are set explicitly to zero. We would therefore expect the optimal parameters to lie in the span of the feature vectors corresponding to the training examples. This is indeed the case.

We will continue with the derivation of the kernel (inner product) form of the linear regression model next time.