Linear models with fixed features

Brooks Paige

Week 3

Linear functions

What is a **linear function**?

$$f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$$
$$cf(\mathbf{x}) = f(c\mathbf{x})$$

Linear functions

What is a **linear function**?

$$f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$$
$$cf(\mathbf{x}) = f(c\mathbf{x})$$

Equation for "a line":

$$f(x) = mx + b$$

Linear functions

What is a **linear function**?

$$f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$$
$$cf(\mathbf{x}) = f(c\mathbf{x})$$

Equation for "a line":

$$f(x) = mx + b$$

Affine function:

$$f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$$

Linear regression

Model: $y_i = \mathbf{w}^\top \phi(\mathbf{x}_i)$

- We have a dataset of input-output examples $(\mathbf{x}_i, y_i), i = 1, \ldots, N$
- We want to learn a mapping from inputs $\mathbf{x}$ to output y such that when we get a new input $\mathbf{x}$, we will produce the correct output.

Linear regression

Model: $y_i = \mathbf{w}^\top \phi(\mathbf{x}_i)$

- We have a dataset of input-output examples $(\mathbf{x}_i, y_i), i = 1, \ldots, N$
- We want to learn a mapping from inputs $\mathbf{x}$ to output y such that when we get a new input $\mathbf{x}$, we will produce the correct output.
- We may have a very large set of such features, only some of which may be relevant for the prediction.

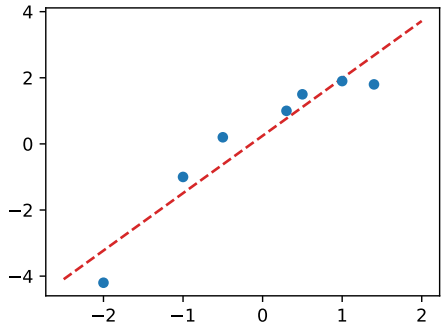
Fitting a line

Feature map for affine functions:

$$\phi(x_i) = [1, x_i]^\top \in \mathbb{R}^D$$

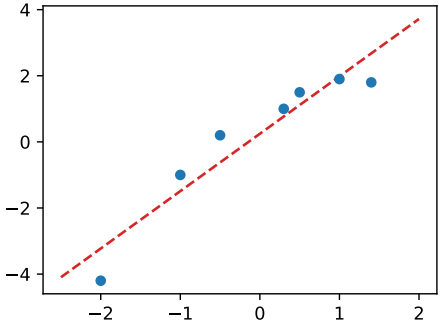
Each of these entries forms a row in a "design matrix"

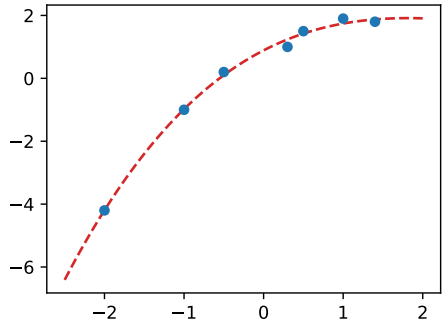
$$\mathbf{\Phi} = \begin{bmatrix} — & \phi(x_1)^\top & — \\ & \vdots & \\ — & \phi(x_N)^\top & — \end{bmatrix} \in \mathbb{R}^{N \times D}.$$



$$\hat{y}_i = 0.25 + 1.74 x_i$$

Fitting a ~~line~~ polynomial





$$\phi(x_i) = [1, x_i]^\top$$

$$\hat{y}_i = 0.25 + 1.74x_i$$

$$\phi(x_i) = [1, x_i, x_i^2, x_i^3]^\top$$

$$\hat{y}_i = 0.89 + 1.31x_i - 0.51x_i^2 + 0.05x_i^3$$

Least squares fit

Assume we want to minimize the squared-error loss

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} (y_i - \mathbf{w}^\top \phi(\mathbf{x}_i))^2.$$

Least squares fit

Assume we want to minimize the squared-error loss

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} (y_i - \mathbf{w}^\top \phi(\mathbf{x}_i))^2.$$

This is one of those rare examples we can actually solve directly! We can simplify notation by defining

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \qquad \mathbf{\Phi} = \begin{bmatrix} — & \phi(x_1)^\top & — \\ & \vdots & \\ — & \phi(x_N)^\top & — \end{bmatrix}$$

and writing in matrix form $\mathcal{L}(\mathbf{w}) = \|\mathbf{y} - \mathbf{\Phi}\mathbf{w}\|_2^2$, where $\|\mathbf{v}\|_2^2 = \mathbf{v}^\top \mathbf{v}$.

Least squares fit

Now, if we take derivatives with respect to $\mathbf{w}$,

$$\frac{\partial}{\partial \mathbf{w}} \mathcal{L}(\mathbf{w}) = \frac{\partial}{\partial \mathbf{w}} (\mathbf{y} - \mathbf{\Phi}\mathbf{w})^\top (\mathbf{y} - \mathbf{\Phi}\mathbf{w})$$

Least squares fit

Now, if we take derivatives with respect to $\mathbf{w}$,

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}} \mathcal{L}(\mathbf{w}) &= \frac{\partial}{\partial \mathbf{w}} (\mathbf{y} - \mathbf{\Phi}\mathbf{w})^\top (\mathbf{y} - \mathbf{\Phi}\mathbf{w}) \\ &= \frac{\partial}{\partial \mathbf{w}} \mathbf{w}^\top \mathbf{\Phi}^\top \mathbf{\Phi}\mathbf{w} - 2\mathbf{w}^\top \mathbf{\Phi}^\top \mathbf{y}\end{aligned}$$

Least squares fit

Now, if we take derivatives with respect to $\mathbf{w}$,

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{w}) &= \frac{\partial}{\partial \mathbf{w}}(\mathbf{y}-\mathbf{\Phi}\mathbf{w})^\top(\mathbf{y}-\mathbf{\Phi}\mathbf{w}) \\ &= \frac{\partial}{\partial \mathbf{w}}\mathbf{w}^\top\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{w} - 2\mathbf{w}^\top\mathbf{\Phi}^\top\mathbf{y} \\ &= 2\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{w} - 2\mathbf{\Phi}^\top\mathbf{y}.\end{aligned}$$

Least squares fit

Now, if we take derivatives with respect to $\mathbf{w}$,

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{w}) &= \frac{\partial}{\partial \mathbf{w}}(\mathbf{y}-\mathbf{\Phi}\mathbf{w})^\top(\mathbf{y}-\mathbf{\Phi}\mathbf{w}) \\ &= \frac{\partial}{\partial \mathbf{w}}\mathbf{w}^\top\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{w} - 2\mathbf{w}^\top\mathbf{\Phi}^\top\mathbf{y} \\ &= 2\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{w} - 2\mathbf{\Phi}^\top\mathbf{y}.\end{aligned}$$

(See **matrix identities**!)

Least squares fit

Now, if we take derivatives with respect to $\mathbf{w}$,

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{w}) &= \frac{\partial}{\partial \mathbf{w}}(\mathbf{y}-\mathbf{\Phi}\mathbf{w})^\top(\mathbf{y}-\mathbf{\Phi}\mathbf{w}) \\ &= \frac{\partial}{\partial \mathbf{w}}\mathbf{w}^\top\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{w} - 2\mathbf{w}^\top\mathbf{\Phi}^\top\mathbf{y} \\ &= 2\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{w} - 2\mathbf{\Phi}^\top\mathbf{y}. \qquad \text{(See matrix identities!)}\end{aligned}$$

Set equal to zero and solve for $\mathbf{w}$ for

$$\mathbf{w} = (\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{y}$$

Least squares fit

Now, if we take derivatives with respect to $\mathbf{w}$,

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{w}) &= \frac{\partial}{\partial \mathbf{w}}(\mathbf{y}-\mathbf{\Phi}\mathbf{w})^\top(\mathbf{y}-\mathbf{\Phi}\mathbf{w}) \\ &= \frac{\partial}{\partial \mathbf{w}}\mathbf{w}^\top\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{w} - 2\mathbf{w}^\top\mathbf{\Phi}^\top\mathbf{y} \\ &= 2\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{w} - 2\mathbf{\Phi}^\top\mathbf{y}. \qquad \text{(See matrix identities!)}\end{aligned}$$

Set equal to zero and solve for $\mathbf{w}$ for

$$\begin{aligned}\mathbf{w} &= (\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{y} \\ &= \left(\sum_{i=1}^{N}\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^\top\right)^{-1}\sum_{i=1}^{N}y_i\phi(\mathbf{x}_i).\end{aligned}$$

Warning!

Our estimator: $\mathbf{w} = (\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{y}$.

Note: $\mathbf{\Phi} \in \mathbb{R}^{N \times D}$. What happens if $D > N$?

Regularization

Regularization

(take 1)

Empirical risk minimization

Reminders:

- we care about **test error**; and
- we assuming training and test data come from the same(-ish) underlying distribution.

The principle of **empirical risk minimization** says: although we cannot minimize the expected error unknown data, we can minimize the empirical error on the known training set.

Empirical risk minimization

Reminders:

- we care about **test error**; and
- we assuming training and test data come from the same(-ish) underlying distribution.

The principle of **empirical risk minimization** says: although we cannot minimize the expected error unknown data, we can minimize the empirical error on the known training set.

This leads to objectives of the form

$$\min \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} \ell(\mathbf{w}^\top \phi(\mathbf{x}_i), y_i)$$

for some per-instance loss $\ell(\cdot, y)$, such as the squared-error loss.

Regularization

As we've seen already, simply minimizing the training loss can lead to overfitting.

Regularization can help. The idea is to add an extra term to the loss, which penalizes over-complicated models, or rapid changes in output:

$$\min \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} \ell(\mathbf{w}^\top \phi(\mathbf{x}_i), y_i) + \lambda r(\mathbf{w}).$$

The function $r(\mathbf{w})$ is called the regularizer.

The parameter λ can be set using a validation set.

L_2 regularization, or "Ridge Regression"

One way you may want to regularize a linear regression model is to prevent large values of the weights by adding a quadratic penalty:

$$\min \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} \underbrace{\left(y_i - \mathbf{w}^\top \phi(\mathbf{x}_i)\right)^2}_{\ell(\mathbf{w}^\top \phi(\mathbf{x}_i), y_i)} + \lambda \underbrace{\mathbf{w}^\top \mathbf{w}}_{r(\mathbf{w})}.$$

Like the standard linear regression model, this also has a closed-form solution (**Exercise**: with similar derivation),

$$\mathbf{w} = (\mathbf{\Phi}^\top \mathbf{\Phi} + \lambda \mathbf{I})^{-1} \mathbf{\Phi}^\top \mathbf{y}.$$

Question: Now what happens if $D > N$?

Redundant features

What happens if you have two *identical* features in your model?

- Obviously you wouldn't intentionally do this, but maybe a column in your data frame has been duplicated (or maybe two columns measure very similar things and are "nearly" identical)

Redundant features

What happens if you have two *identical* features in your model?

- Obviously you wouldn't intentionally do this, but maybe a column in your data frame has been duplicated (or maybe two columns measure very similar things and are "nearly" identical)
- Simple linear regression: **Singular matrix** — $(\mathbf{\Phi}^\top \mathbf{\Phi})$ is rank-deficient
  - (the loss $(y_i - w_1 x_{i,1} - w_2 x_{i,2})^2$ is the same for any values w_1, w_2 with the same sum $w_1 + w_2$)

Redundant features

What happens if you have two *identical* features in your model?

- Obviously you wouldn't intentionally do this, but maybe a column in your data frame has been duplicated (or maybe two columns measure very similar things and are "nearly" identical)
- Simple linear regression: **Singular matrix** — $(\mathbf{\Phi}^\top \mathbf{\Phi})$ is rank-deficient
  - (the loss $(y_i - w_1 x_{i,1} - w_2 x_{i,2})^2$ is the same for any values w_1, w_2 with the same sum $w_1 + w_2$)
- Ridge regression: both have equal weights, $w_1 = w_2$
  - (for fixed $w_1 + w_2$, then $w_1^2 + w_2^2$ is minimized when $w_1 = w_2$)

Redundant features

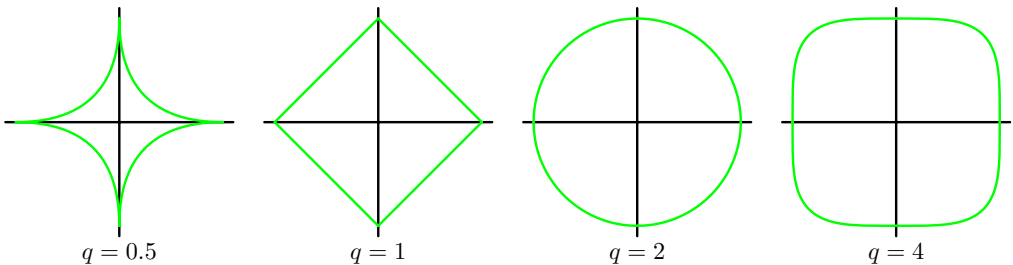
What happens if you have two *identical* features in your model?

- Obviously you wouldn't intentionally do this, but maybe a column in your data frame has been duplicated (or maybe two columns measure very similar things and are "nearly" identical)
- Simple linear regression: **Singular matrix** — $(\mathbf{\Phi}^\top \mathbf{\Phi})$ is rank-deficient
  - (the loss $(y_i - w_1 x_{i,1} - w_2 x_{i,2})^2$ is the same for any values w_1, w_2 with the same sum $w_1 + w_2$)
- Ridge regression: both have equal weights, $w_1 = w_2$
  - (for fixed $w_1 + w_2$, then $w_1^2 + w_2^2$ is minimized when $w_1 = w_2$)

Another goal might be a **sparse** solution: select only one or the other!

Different penalty terms

Consider different regularizers $r(\mathbf{w}) = \sum_{d=1}^{D} |w_d|^q$, parameterized by q.



- As $q \to 0$, we approach a "counting" regularizer, i.e. # of nonzero entries
- The objective function is complicated (non convex) for $q < 1$

Figure: Bishop Ch. 3

L_1 versus L_2 regularization

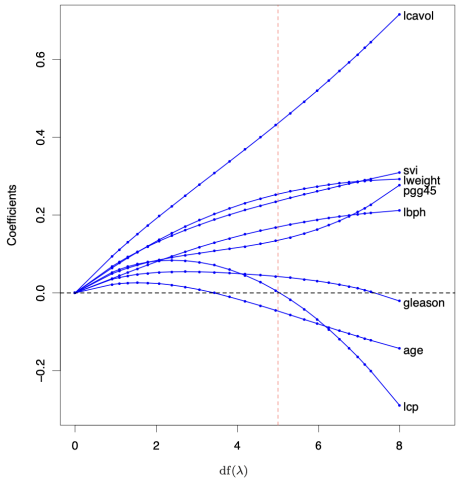
$L_2(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_d w_d^2$

- Sum of squared weights
- Heavily penalizes large w_d but only slightly penalizes small w_d
- Differentiable; closed-form solution
- Small weights still persist (don't reach exactly zero)

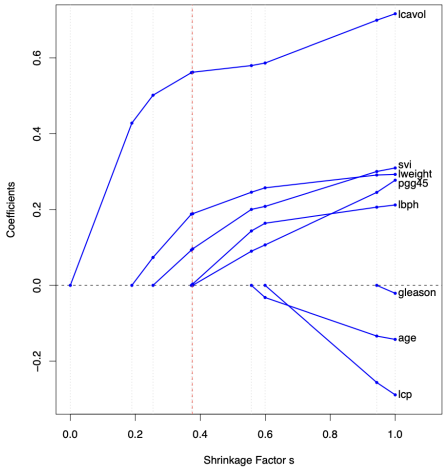
$L_1(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_d |w_d|$

- Sum of the absolute values of the weights
- Non-differentiable at zero; need to use specialized optimization routines
- Only a subset of weights remain — redundant features will be "pruned" to zero

Comparing solutions with varying λ



Ridge (L_2) regularization



LASSO (L_1) regularization

Fig. 3.8 & 3.10, ESL (Hastie, Tibshirani, & Friedman)

Probabilistic treatment

Maximum likelihood and MAP estimation (1/3)

Recall our "general" regularized loss:

$$\min \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} \ell(\mathbf{w}^\top \phi(\mathbf{x}_i), y_i) + \lambda r(\mathbf{w}).$$

With a squared error loss, and an L_2 regularizer, this then becomes

$$\min \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^\top \phi(\mathbf{x}_i) - y_i)^2 + \lambda \mathbf{w}^\top \mathbf{w}.$$

Maximum likelihood and MAP estimation (2/3)

Now consider an alternative problem maximum likelihood estimation or MAP estimation in fitting a probabilistic model of the form

$$p(\mathbf{w}, y_1, \ldots, y_N | \mathbf{x}_1, \ldots, \mathbf{x}_N) = p(\mathbf{w}) \prod_{i=1}^{N} p(y_i | \mathbf{x}_i, \mathbf{w}).$$

Maximizing the log likelihood, i.e. ignoring the prior $p(\mathbf{w})$, means finding

$$\hat{\mathbf{w}}^{ML} = \arg\max_{\mathbf{w}} \sum \log p(y_i | \mathbf{x}_i, \mathbf{w})$$

and the MAP estimate would be

$$\hat{\mathbf{w}}^{MAP} = \arg\max_{\mathbf{w}} \sum \log p(y_i | \mathbf{x}_i, \mathbf{w}) + \log p(\mathbf{w}).$$

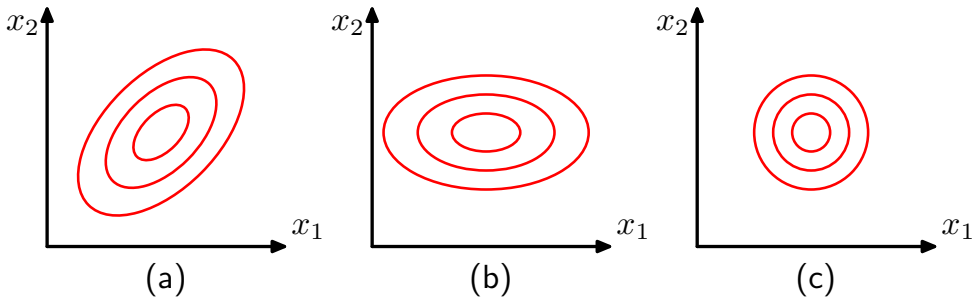
Aside: Multivariate Gaussian distributions

$$\log p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{D}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$$

Figure: Bishop

Aside: Multivariate Gaussian distributions

$$\log p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{D}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$$



$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix} \qquad \boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_D^2) \qquad \boldsymbol{\Sigma} = \sigma^2\mathbf{I}$$

Figure: Bishop

Gaussian priors and likelihoods

Let $y_i \sim \mathcal{N}(\mathbf{w}^\top \phi(\mathbf{x}_i), \sigma^2)$: that is, suppose labels y_i have a Gaussian likelihood with mean $\mathbf{w}^\top \phi(\mathbf{x}_i)$ with some "error" that has (known) variance σ^2. Then

$$\log p(y_i|\mathbf{x}_i, \mathbf{w}) = \underbrace{-\frac{1}{2}\log 2\pi\sigma^2}_{\text{const.}} - \frac{1}{2\sigma^2}(y - \mathbf{w}^\top \phi(\mathbf{x}))^2.$$

Gaussian priors and likelihoods

Let $y_i \sim \mathcal{N}(\mathbf{w}^\top \phi(\mathbf{x}_i), \sigma^2)$: that is, suppose labels y_i have a Gaussian likelihood with mean $\mathbf{w}^\top \phi(\mathbf{x}_i)$ with some "error" that has (known) variance σ^2. Then

$$\log p(y_i|\mathbf{x}_i, \mathbf{w}) = \underbrace{-\frac{1}{2}\log 2\pi\sigma^2}_{\text{const.}} - \frac{1}{2\sigma^2}(y - \mathbf{w}^\top \phi(\mathbf{x}))^2.$$

Now suppose we place a zero-mean Gaussian prior on $\mathbf{w}$, with a isotropic covariance $\mathbf{\Sigma} = \alpha^{-1}\mathbf{I}$

$$\log p(\mathbf{w}|\alpha) = \underbrace{-\frac{D}{2}\log 2\pi + \frac{D}{2}\log \alpha}_{\text{const.}} - \frac{\alpha}{2}\mathbf{w}^\top \mathbf{w}.$$

Maximum likelihood and MAP estimation (3/3)

With Gaussian priors, dropping terms which do not depend on $\mathbf{w}$, ML and MAP estimates correspond to

$$\hat{\mathbf{w}}^{ML} = \arg\max_{\mathbf{w}} -\frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \mathbf{w}^\top\phi(\mathbf{x}_i))^2$$

$$\hat{\mathbf{w}}^{MAP} = \arg\max_{\mathbf{w}} -\frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \mathbf{w}^\top\phi(\mathbf{x}_i))^2 - \frac{\alpha}{2}\mathbf{w}^\top\mathbf{w}.$$

Maximum likelihood and MAP estimation (3/3)

With Gaussian priors, dropping terms which do not depend on $\mathbf{w}$, ML and MAP estimates correspond to

$$\hat{\mathbf{w}}^{ML} = \arg\max_{\mathbf{w}} -\frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \mathbf{w}^\top\phi(\mathbf{x}_i))^2$$

$$\hat{\mathbf{w}}^{MAP} = \arg\max_{\mathbf{w}} -\frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \mathbf{w}^\top\phi(\mathbf{x}_i))^2 - \frac{\alpha}{2}\mathbf{w}^\top\mathbf{w}.$$

Flipping back to the previous slide, you can see that **the MAP objective corresponds to the regularized empirical risk objective**, with an L_2 regularizer, where $\lambda \equiv \alpha\sigma^2$.

Laplace distribution

There's a similar equivalency for L_1 regularization, by placing a **Laplace distribution** prior on each w_d:

$$p(w_d|\mu, b) = \frac{1}{2b} \exp\left\{-\frac{|w_d - \mu|}{b}\right\}$$

This kinda looks like a normal distribution, but has heavier tails.

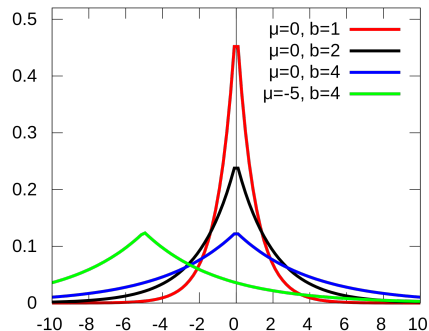


Figure: wikipedia

Fully Bayesian treatment?

Bayesian linear regression

Reminder: MAP estimation is not Bayesian!

Bayesian linear regression

Reminder: MAP estimation is not Bayesian!

As it turns out, Bayesian linear regression is one of the only models where the posterior has a closed form. If we define the model as

$$\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \alpha^{-1}\mathbf{I}_d) \qquad\qquad y_i \sim \mathcal{N}(\mathbf{w}^\top \phi(\mathbf{x}_i), \beta^{-1})$$

then the posterior is also a normal distribution

$$p(\mathbf{w}|\mathbf{x}_{1:N}, y_{1:N}) = \mathcal{N}(\mathbf{w}|\mathbf{m}, \mathbf{S})$$

where

$$\mathbf{m} = \beta(\alpha\mathbf{I} + \beta\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{y}, \qquad\qquad \mathbf{S} = (\alpha\mathbf{I} + \beta\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}.$$

More on Bayesian linear regression

- Note that the posterior mean $\mathbf{m}$ was the same as the ridge regression solution!
- This only worked out nicely because everything was Gaussian. If you would like to convince yourself of this result, you can show it using the Gaussian identities in the "cheat sheets" posted with the week 1 material, or Bishop PRML Appendix B.
- Other priors and other likelihoods will require more work — we'll come back to this.

Linear classification

What about linear models for classification?

We can also apply the regularized learning framework

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} \ell(\mathbf{w}^\top \phi(\mathbf{x}_i), y_i) + \lambda r(\mathbf{w})$$

to classification problems. It just involves changing ℓ!

Binary classification: Bernoulli likelihood

Maximum likelihood estimation:

- Define $f(\mathbf{x}_i, \mathbf{w})$ which returns a value in the range $[0, 1]$
- Interpret this output as a probability $p(y_i = 1|\mathbf{x}_i, \mathbf{w})$
- Optimize the weights $\mathbf{w}$ by maximizing the (log-)probability of the labels given the data:

$$\begin{aligned}\hat{\mathbf{w}} &= \arg\max_{\mathbf{w}} \sum_{i=1}^{N} \log p(y_i = 1|\mathbf{x}_i, \mathbf{w}) \\ &= \arg\max_{\mathbf{w}} \sum_{i=1}^{N} y_i \log f(\mathbf{x}_i, \mathbf{w}) + (1 - y_i) \log(1 - f(\mathbf{x}_i, \mathbf{w}))\end{aligned}$$

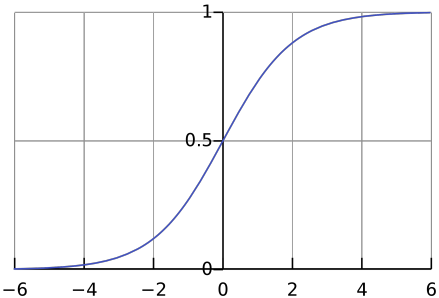
This (with flipped sign) is also called **binary cross-entropy** loss.

Sigmoid functions

We can "squash" a real value into the interval $(0, 1)$ using a sigmoid function.

The logistic function is a classic choice,

$$\text{logistic}(z) = \frac{1}{1 + e^{-z}}.$$



This gives a per-datapoint likelihood

$$p(y_i = 1|\mathbf{x}_i, \mathbf{w}) = \frac{\exp\{\mathbf{w}^\top \phi(\mathbf{x}_i)\}}{1 + \exp\{\mathbf{w}^\top \phi(\mathbf{x}_i)\}} = \frac{1}{1 + \exp\{-\mathbf{w}^\top \phi(\mathbf{x}_i)\}}.$$

Figure: wikipedia

Multi-class classification (1/2)

For multi-class classification, assume we have K classes.

The probabilistic approach is to choose a likelihood which defines a categorical distribution over those K classes. This is typically done with the **softmax function**.

- We need a one-hot encoding of y_i, i.e. $\mathbf{y}_i$ is a vector of length K with a single non-zero entry, with $y_{i,k} = 1$ for the observed class k.
- We need a matrix $\mathbf{W} \in \mathbb{R}^{K \times D}$, in order to output all K values of $\mathbf{y}_i$.

Multi-class classification (2/2)

Define **logits**, unnormalized log probabilities, as a vector $\mathbf{z}_i \in \mathbb{R}^K$, with

$$\mathbf{z}_i = \mathbf{W}^\top \phi(\mathbf{x}_i).$$

The softmax function computes a probability vector $\hat{\mathbf{y}}_i$ over the different classes,

$$\hat{y}_{i,k} = \frac{\exp(z_{i,k})}{\sum_{j=1}^{K} \exp(z_{i,j})}.$$

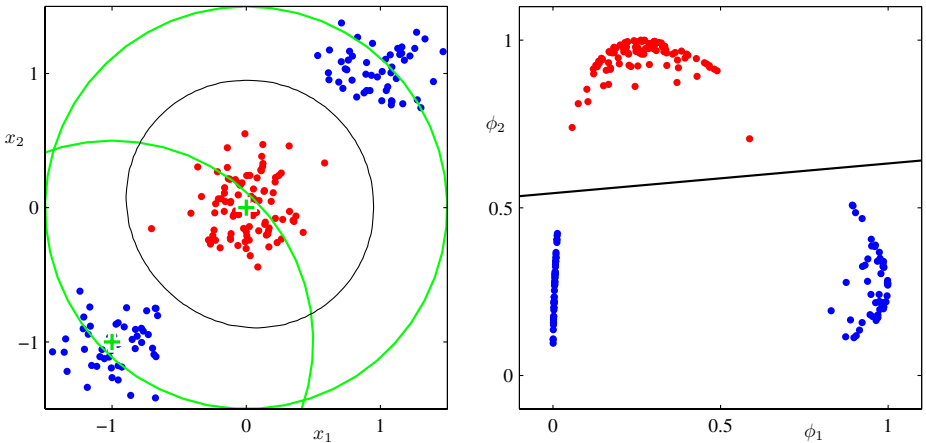
The log-likelihood of a discrete probability distribution yields the loss

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\sum_{j=1}^{K} y_{i,j} \log \hat{y}_{i,j}.$$

Exercise: convince yourself that for $K = 2$ this is identical to logistic regression.

What should the features be?

Example: two-class classification



Original space is on the left. The transformed space has two features $\phi_1(\mathbf{x}) = \exp(-(\mathbf{x}+1)^2/2)$ and $\phi_2(\mathbf{x}) = \exp(-\mathbf{x}^2/2)$.

Figure: Bishop PRML, Figure 4.12

Performance depends on the features

How well your linear learner works depends on the input features ϕ.

- If you have lots of inputs $\mathbf{x}$, and they are fairly "useful", you'll likely do well
- If you have good inputs, but your target variable isn't linear in $\mathbf{x}$, define appropriate transforms ϕ

What if you have no idea how to construct ϕ, and $\mathbf{x}$ on its own isn't good enough?

- **Kernel methods**, which create large (potentially infinite) numbers of implicit features
- **Deep learning**, where we learn appropriate feature maps ϕ by gradient descent

Performance depends on the features

How well your linear learner works depends on the input features ϕ.

- If you have lots of inputs $\mathbf{x}$, and they are fairly "useful", you'll likely do well
- If you have good inputs, but your target variable isn't linear in $\mathbf{x}$, define appropriate transforms ϕ

What if you have no idea how to construct ϕ, and $\mathbf{x}$ on its own isn't good enough?

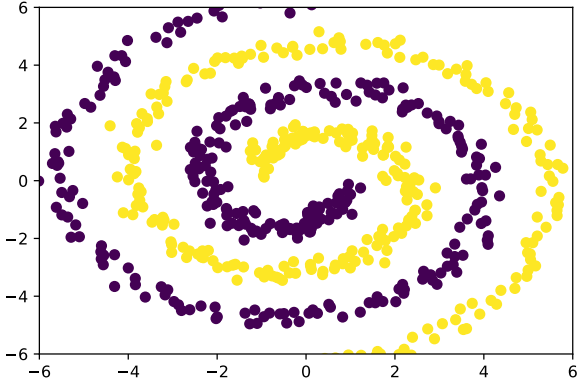
- **Kernel methods**, which create large (potentially infinite) numbers of implicit features
- **Deep learning**, where we learn appropriate feature maps ϕ by gradient descent

First, though, a fun end-of-lecture example: **random features**.

Spiral dataset

Here's a 2d binary classification dataset.

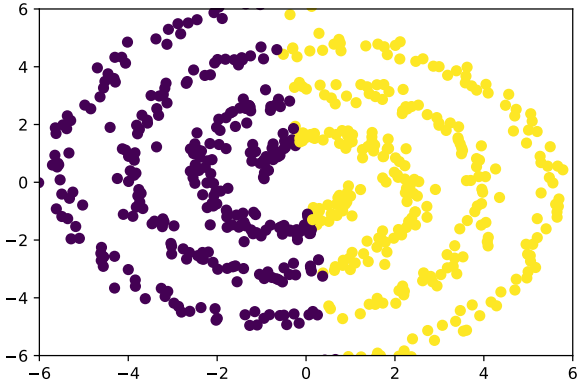
- Classes correspond to two arms of a spiral
- Clearly, this cannot be classified just with the two coordinates and an intercept!



Spiral dataset

This is the result of fitting a logistic regression model with

- L_2 regularizer
- $\phi(\mathbf{x}) = [1, x_1, x_2]$

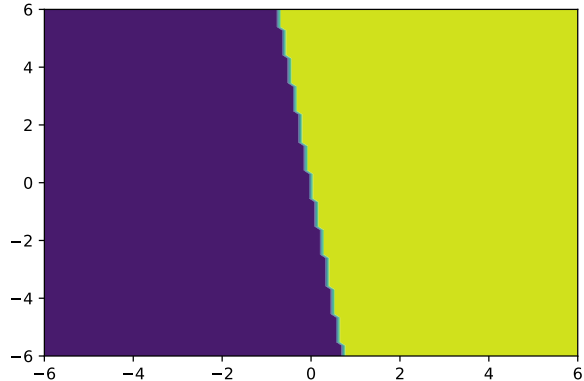


Spiral dataset

This is the result of fitting a logistic regression model with

- L_2 regularizer
- $\phi(\mathbf{x}) = [1, x_1, x_2]$

Here's the decision boundary.



Random features

Any idea what a good choice would be for $\phi(\mathbf{x})$?

Random features

Any idea what a good choice would be for $\phi(\mathbf{x})$? One surprisingly versatile option is to use random cosine features. Suppose our original data $\mathbf{x} \in \mathbb{R}^M$, and we want to construct D features. Let

$$\phi(\mathbf{x}) = \cos(\mathbf{A}\mathbf{x} + \mathbf{b})$$

where $\mathbf{A} \in \mathbb{R}^{D \times M}$ and $\mathbf{b} \in \mathbb{R}^D$ are chosen at random (and then held fixed), with each entry

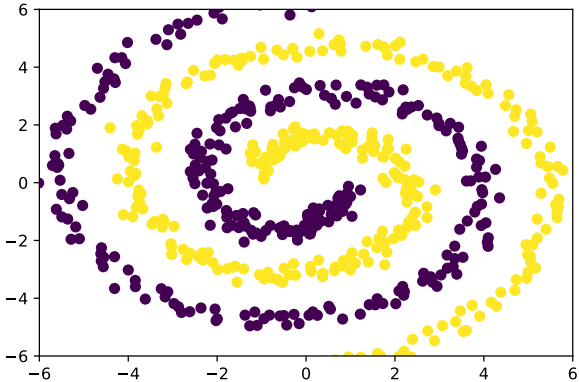
$$\begin{aligned} A_{ij} &\sim \mathcal{N}(0, 1) \\ b_i &\sim \text{Uniform}([0, 2\pi]). \end{aligned}$$

Note that we can have $D \gg M$, to create very large random feature spaces!

Random features for the spiral dataset

This is the result of fitting a logistic regression model with

- L_2 regularizer
- $\phi(\mathbf{x}) = \cos(\mathbf{A}\mathbf{x} + \mathbf{b})$, plus intercept
- 80 random features



Random features for the spiral dataset

This is the result of fitting a logistic regression model with

- L_2 regularizer
- $\phi(\mathbf{x}) = \cos(\mathbf{A}\mathbf{x} + \mathbf{b})$, plus intercept
- 80 random features

Here's the decision boundary!