Linear models and regularization

Brooks Paige

Week 2

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.

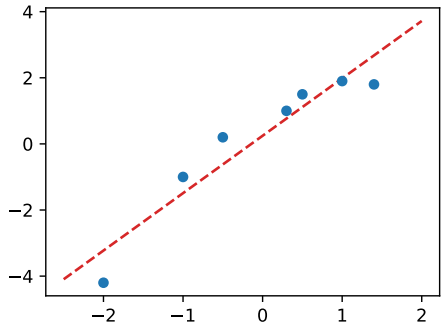
Fitting a line

Feature map for affine functions:

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

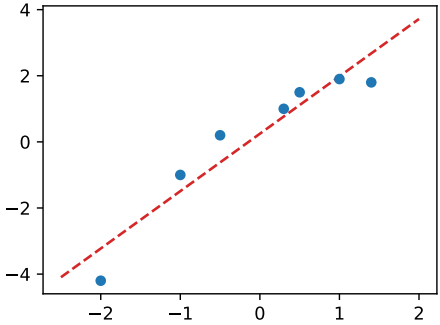
Each of these entries forms a row in a "design matrix"

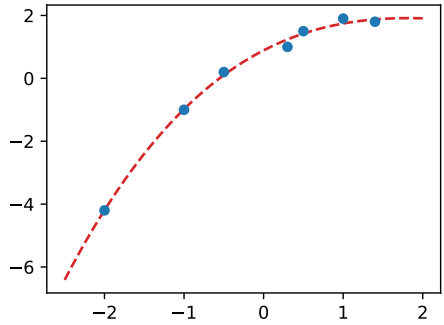
$$\mathbf{\Phi} = \begin{bmatrix} — & \phi(x_1)^\top & — \\ & \vdots & \\ — & \phi(x_N)^\top & — \end{bmatrix} \in \mathbb{R}^{N \times D}.$$



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

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

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. We will look at a few examples.

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 (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

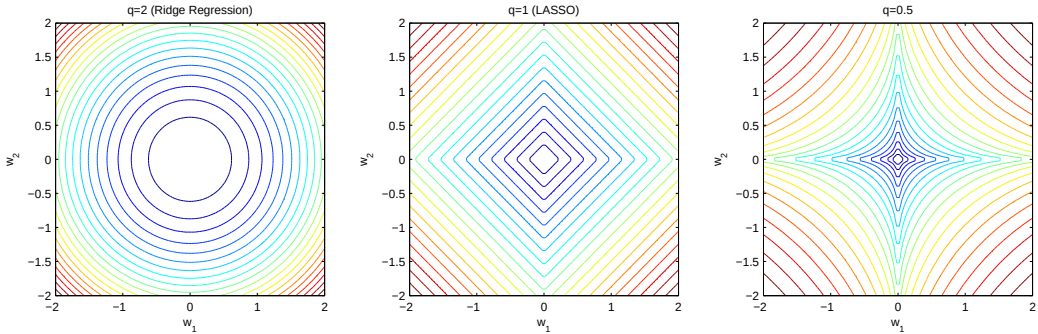
What happens if you have two *identical* features in your model?

- 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: David Barber

L_1 versus L_2 regularization

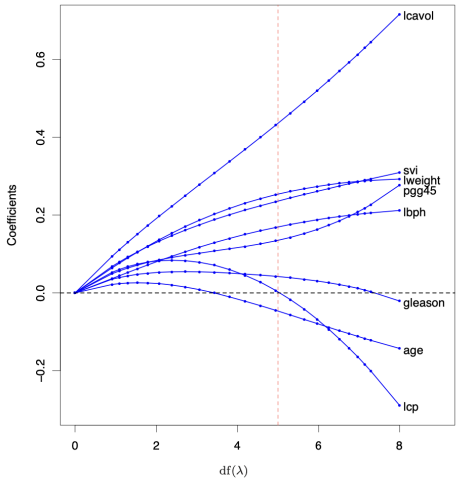
$L_2(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_i w_i^2$

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

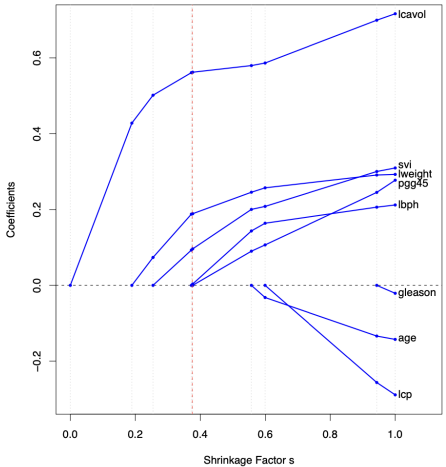
$L_1(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_i |w_i|$

- 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)

Can this replace “manual” feature selection?

Sort of!

Can this replace "manual" feature selection?

Sort of!

- Classical approach: ask a domain expert which features are likely to be important
- LASSO idea: throw everything in the model, see what sticks

Question: Do you care about the coefficients?

Can this replace "manual" feature selection?

Sort of!

- Classical approach: ask a domain expert which features are likely to be important
- LASSO idea: throw everything in the model, see what sticks

Question: Do you care about the coefficients?

- For highly-correlated features, only one of them will likely be included in the final model
- . . . but, both of them independently are good predictors!
- Be careful not to interpret $w_d = 0$ as "feature d is unimportant"

Linear classification

What about linear models for classification?

We can also apply our 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 losses

Let's assume we have binary labels for positive and negative classes, i.e.

$$y_i \in \{+1, -1\}.$$

- Logistic loss:

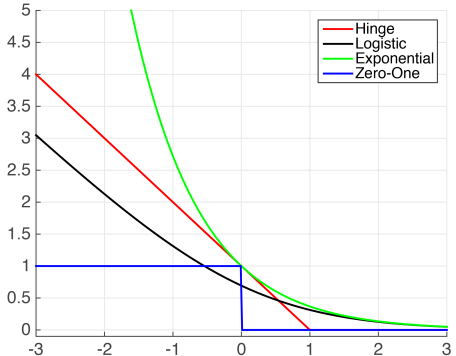
$$\ell(\mathbf{w}^\top \phi(\mathbf{x}_i), y_i) = \log(1 + \exp(-\mathbf{w}^\top \phi(\mathbf{x}_i) y_i)$$

- Hinge loss:

$$\ell(\mathbf{w}^\top \phi(\mathbf{x}_i), y_i) = \max(1 - \mathbf{w}^\top \phi(\mathbf{x}_i) y_i, 0)$$

We talked about the logistic loss last week, but the hinge loss is a bit different!

Binary classification losses

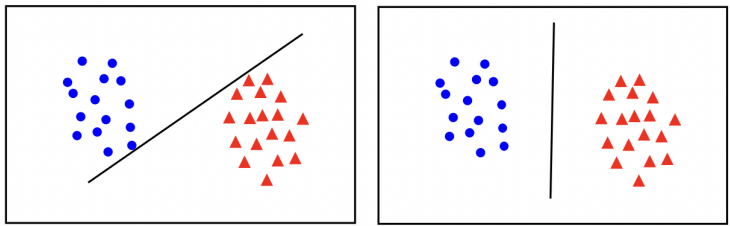


X-axis of plot: $\mathbf{w}^\top \phi(\mathbf{x}) y$

Figure: Kilian Weinberger

Support vector machine and hinge loss

The hinge loss is used in a linear **support vector machine**.



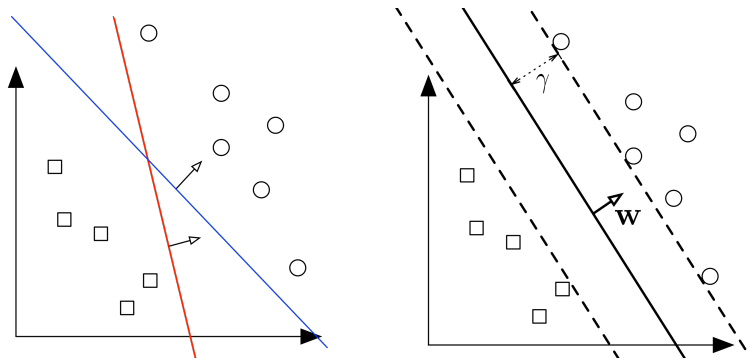
The motivation here is that for data which is linearly separable, there are many separating hyperplanes. How should you choose?

One idea is to maximize the **margin** between the class instances.

Figure: Andrew Zisserman

Support vector machine and hinge loss

The **margin** γ is the shortest-line distance from the nearest datapoint to the separating hyperplane.



The data points which are at the margin are the **support vectors**.

Figure: Kilian Weinberger

Support vector machine and hinge loss

Leaving out the details, solving the problem of finding the max-margin hyperplane involves finding

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \max(1 - \mathbf{w}^\top \phi(\mathbf{x}_i) y_i, 0) + \lambda \|\mathbf{w}\|_2^2.$$

This objective is typically derived by solving a soft-constrained optimization problem. However, it also fits into our framework!

Support vector machine and hinge loss

Leaving out the details, solving the problem of finding the max-margin hyperplane involves finding

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \underbrace{\max(1 - \mathbf{w}^\top \phi(\mathbf{x}_i) y_i, 0)}_{\ell(\dots)} + \lambda \underbrace{\|\mathbf{w}\|_2^2}_{r(\mathbf{w})}.$$

This objective is typically derived by solving a soft-constrained optimization problem. However, it also fits into our framework!

- Hinge loss $\ell(\mathbf{w}^\top \phi(\mathbf{x}), y)$
- L_2 regularizer $r(\mathbf{w})$

Multi-class classification

How do we handle multi-class classification? Assume we have K classes.

The easiest way that is still occasionally useful is to train K classifiers — **one vs. rest**. Each classifier predicts whether the instance $\mathbf{x}_i$ is of a particular class k, or not (i.e. it is one of the $K-1$ other classes).

It's simple and straightforward, but has problems:

- It's expensive, particularly as K gets large;
- It can be incoherent or ambiguous, e.g. if multiple classifiers all give a positive prediction.

Multi-class classification

For true multi-class classification, we can use the **softmax function** and make a small extension to our framework.

- 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 $\mathbf{y}_i$.

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

Recap so far

Many (linear) learning problems can be framed as

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

Mix and match different losses and regularizers! Important cases:

- **Regression losses**: Squared error loss, absolute error loss
- **Classification losses**: Log-loss, hinge loss
- **Regularizers**: L_2 (shrinkage), L_1 (sparsity)

Not nearly exhaustive! Can be selected based on problem requirements.

How useful are linear models?

Feature engineering

Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.

– Andrew Ng (attributed)

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

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**, which is basically the study of automatically learning appropriate feature maps ϕ

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?

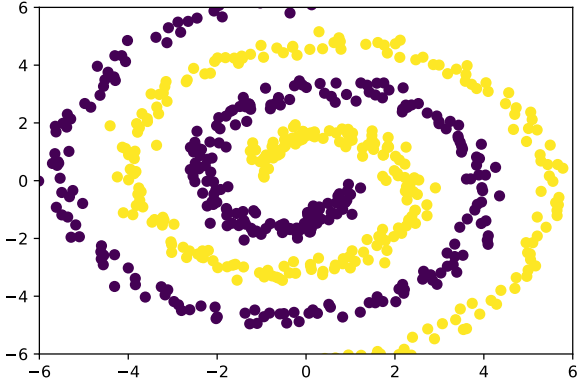
- **Kernel methods**, which create large (potentially infinite) numbers of implicit features
- **Deep learning**, which is basically the study of automatically learning appropriate feature maps ϕ

Before that though I want to leave you with one alternative: **random features**.

Spiral dataset

Here's a 2d binary classification dataset.

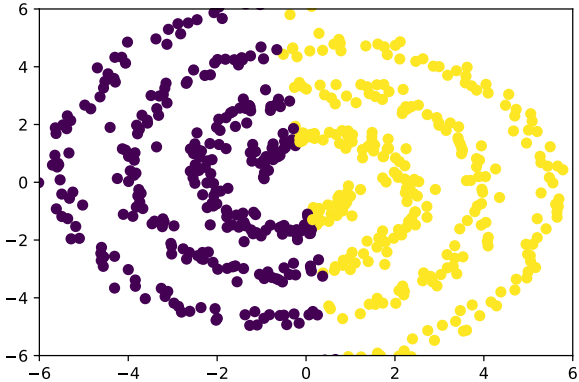
- Classes correspond to two arms of a spiral
- Clearly, this cannot be classified just with the two coordinates and an intercept!



Spiral dataset

This is the result of fitting a logistic regression model with

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

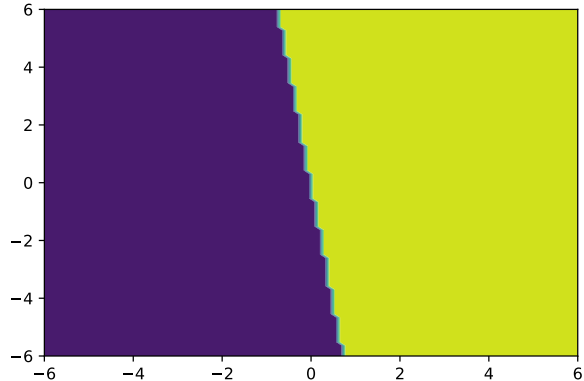


Spiral dataset

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

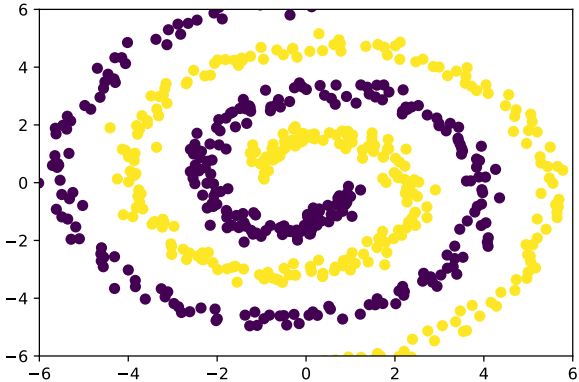
$$\begin{aligned} A_{ij} &\sim \mathcal{N}(0, 1) \\ b_i &\sim \text{Uniform}([0, 2\pi]). \end{aligned}$$

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

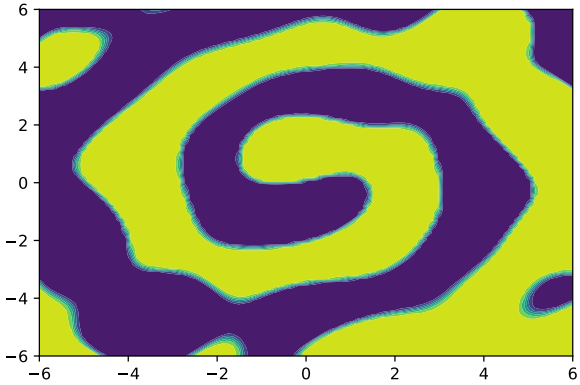


Random features for the spiral dataset

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!



That's it for linear models!