Designing, Visualizing and Understanding Deep Neural Networks

Lecture 3: Machine Learning Background II

CS 182/282A Spring 2020
John Canny

Last Time

- Generative vs. Discriminative models:
  - Deep nets are discriminative models.
  - May learn more slowly at first, but better asymptotic accuracy.

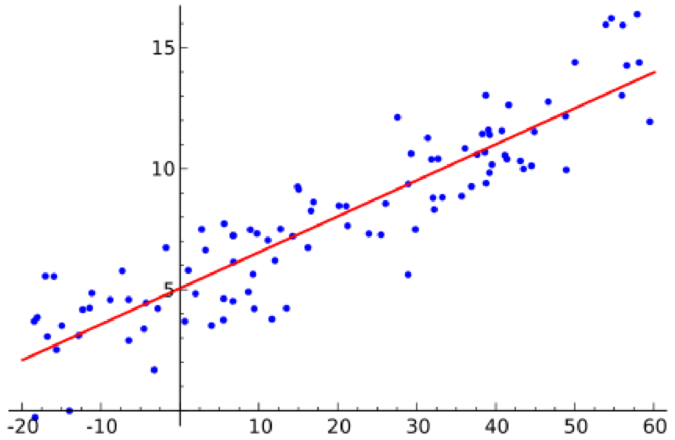
Generative:	**Discriminative:**
Noisy linear functions	Linear Least Squares
Naïve Bayes	Logistic Regression
Hidden Markov Models	Conditional Random Fields
Gaussian mixture models	
Latent Dirichlet Allocation	
	Support Vector Machines (SVM)
	Decision trees + Random Forests
	Neural Networks

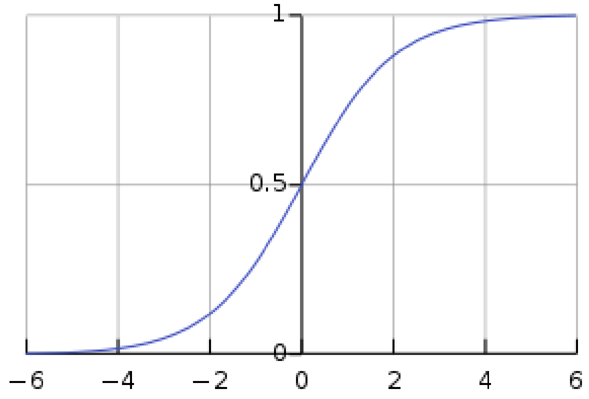
Last Time

- Loss: a measure of difference between predictions $\hat{y}$ and actual targets y.
- A good model should minimize loss on its predictions.
- Risk:
  - Risk is expected loss (so applies to new data)
  - Empirical risk is the loss on a finite subset of data.
- Machine learning models seek to minimize risk, but usually minimize empirical risk.

Last Time

- Prediction functions $\hat{y} = f(x)$:
  - Linear regression: predict a real value, loss = squared loss.
  - Logistic regression: predict a binary target, loss = cross-entropy loss.





Updates – Assignment 1

Assignment 1 is out now, due on Monday Feb 17th, 11pm.

- Uses python + ipython. Please check right away that you have a working installation of python 3 and ipython, and that you can load and execute the assignment notebooks.
- Python virtualenv is your best bet, but there are tricks to doing it on a Mac.

- The assignment itself closely tracks the lecture material over the next couple of weeks, so you'll get best value by doing it in stages.

This Time

- Bias-Variance tradeoff
- Regularization
- Support Vector Machines (SVMs).
- Multi-class Logistic Regression and Softmax.
- Cross-validation.

Bias and Variance

A model $Y = \hat{f}(X)$ is a statistical estimate of a true function $Y = f(X)$.

Because of this, its subject to bias and variance:

Bias: if we train models $\hat{f}_D(X)$ on different datasets D, bias at a point x is the expected difference between their predictions and the true y.

i.e.
$$Bias\left(\hat{f}(x)\right) = \mathrm{E}\left[\hat{f}_D(x) - f(x)\right] = \bar{f}(x) - f(x)$$

The expectation is taken over the training datasets D used to train each model, and $\bar{f}(x) = \mathrm{E}\left[\hat{f}_D(x)\right]$ is the expected prediction at x

Variance: if we train models $\hat{f}_D(X)$ over different datasets D, variance is the variance of the predictions:

$$Variance\left(\hat{f}(x)\right) = \mathrm{E}\left[\left(\hat{f}_D(x) - \bar{f}(x)\right)^2\right]$$

Aside: Expected Values

For a function $f(X)$ an expected value tells us what we expect to see if we sample X from a distribution.

Finite case (m values for X):

$$E[f(X)] = \sum_{i=1}^{m} f(X_i)P(X_i)$$

Continuous case (n-dimensional) $X = (X_1, \dots, X_n)$:

$$E[f(X)] = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f(X)P(X)dX_1 \dots dX_n$$

Aside: Expected Value Notation

Sometimes the expectation is taken over something other than the main argument to $f(X)$ in which case its important to specify the variable:

$$E_D[f_D(X)]$$

Where for Bias and Variance estimates, D is a training dataset.

Better still, one specifies the sampling variable and its distribution

$$E_{D \sim P(D)}[f(X)]$$

Where $P(D)$ is the probability or probability density of the training dataset D.

The notation $D \sim P(D)$ means that D is sampled from the distribution $P(D)$.

Bias-Variance Decomposition

For squared-loss problems the total squared error decomposes into bias and variance:

$$\mathrm{E}_D\left[\left(\hat{f}_D(x) - f(x)\right)^2\right] = \mathrm{E}_D\left[\left(\hat{f}_D(x) - \bar{f}(x) + \bar{f}(x) - f(x)\right)^2\right]$$

Expected values (by *D*) of $\hat{f}(x)$, so constant

$$= \mathrm{E}_D\left[\left(\underbrace{\hat{f}_D(x) - \bar{f}(x)}_{\text{Variance contribution}} + \underbrace{Bias}_{\text{Bias contribution}}\right)^2\right]$$

$$= \mathrm{E}_D\left[\left(\hat{f}_D(x) - \bar{f}(x)\right)^2\right] + Bias^2 - \cancel{2\mathrm{E}_D\left[\left(\hat{f}_D(x) - \bar{f}(x)\right) Bias\right]}$$

$$= Variance\left(\hat{f}_D(x)\right) + Bias\left(\hat{f}_D(x)\right)^2$$

Bias-Variance Tradeoff

The total expected error is

$$Bias^2 + Variance$$

Because of the bias-variance trade-off, we want to **balance** these two contributions.

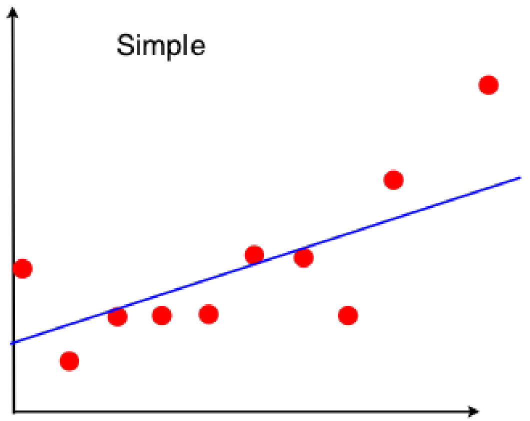
If $Variance \gg Bias^2$, it means there is too much variation between models. This is called **over-fitting** .

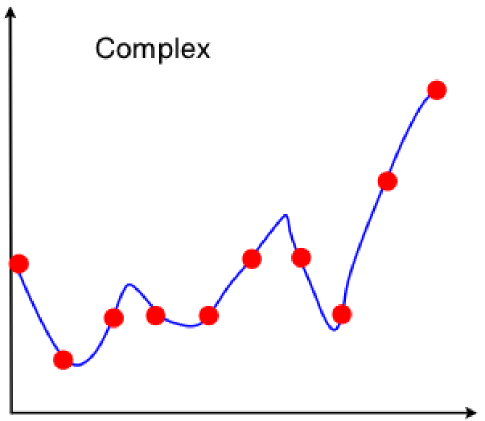
If $Bias^2 \gg Variance$, then the models are not fitting the data well enough. This is called **under-fitting** .

Bias-Variance Tradeoff

A linear model can only fit a straight line, so has **high bias** . Linear models fit to different datasets vary only slightly, so they have **low variance** .

A high-degree polynomial can fit a complex curve, so **low bias** . But the polynomial tends to overfit the data sample, leading to **high variance** .

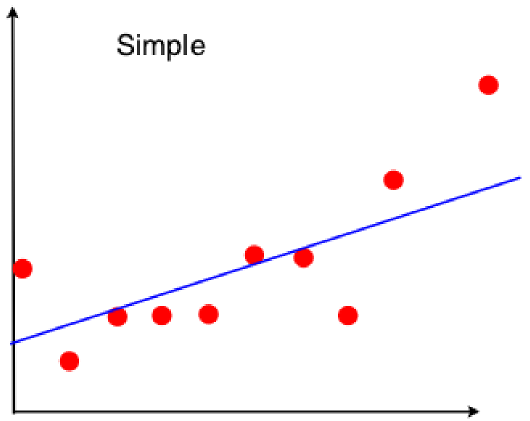


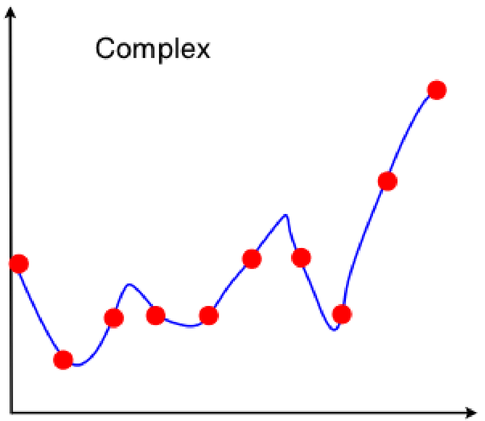


Bias-Variance Tradeoff

Deep Networks have high-dimensional parametrizations so generally live on the right side graph. They tend to have high variance and low bias compared to other models.

Variance in models can be reduced by regularization, and this is an important theme in the design of deep networks.





Regularization

Multivariate linear regression: $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^k$, and the model is

$$y = Ax$$

for a $k \times m$ matrix A (assume $x_m = 1$ so no constant term is needed).

The MSE (Minimum Square Error) model from last lecture is

$$\hat{A} = M_{yx} M_{xx}^{-1}$$

where $M_{xx} = \sum_{i=1}^{n} x_i x_i^T$ and $M_{yx} = \sum_{i=1}^{n} y_i x_i^T$

But what if M_{xx} is singular, or nearly so?

Linear Algebra

A matrix M is positive definite iff

A. $v^T M v \geq 0$ for all real v

B. $v^T M v < 0$ for all real v

C. $v^T M v > 0$ for all real v

D. $v^T M v \leq 0$ for all real v

Oops

A matrix M is **positive definite** iff

A. ~~$v^T M v \geq 0$ for all real v~~

Try Again

Continue

Oops

A matrix M is positive definite iff

B. ~~$v^T M v < 0$ for all real v~~

Try Again

Continue

Yes!

A matrix M is positive definite iff

C. $v^T M v > 0$ for all real v

Try Again

Continue

Oops

A matrix M is positive definite iff

D. ~~$v^T M v \leq 0$ for all real v~~

Try Again

Continue

Linear Algebra

A matrix M is **positive semi -definite** iff

$v^T M v \geq 0$ for all real vectors v.

Now for $M_{xx} = \sum_{i=1}^{n} x_i x_i^T$, we have

$$v^T M_{xx} v = \sum_{i=1}^{n} v^T x_i x_i^T v = \sum_{i=1}^{n} \left(v^T x_i\right)^2 \geq 0$$

which is a sum of squares, so M_{xx} is positive semi-definite.

Linear Algebra

For a matrix M, an eigenvector v_i is a vector such that $Mv_i = \lambda_i v_i$, for a scalar λ_i, which is the corresponding eigenvalue .

A positive semi-definite matrix M has all real, non-negative eigenvalues.

It has an eigendecomposition as:

$$M = UDU^T$$

Where U is a unitary matrix ($U^{-1} = U^T$) whose columns are the eigenvectors of M, and D is a diagonal matrix containing the eigenvalues of M. i.e.

$$U = [v_1 \quad v_2 \quad \dots \quad v_m] \qquad D = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_m \end{bmatrix}$$

Linear Algebra

If M is non-singular, the inverse of $M = UDU^T$ is $M^{-1} = UD^{-1}U^T$ where:

$$D^{-1} = \begin{bmatrix} \lambda_1^{-1} & 0 & \dots & 0 \\ 0 & \lambda_2^{-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_m^{-1} \end{bmatrix}$$

M is non-singular is equivalent to $\lambda_i > 0$ for all i.

Even a few small λ_i of M_{xx} causes high variance in models across training datasets because of the inverse of M_{xx} in the regression model derivation.

Can we force the λ_i to be sufficiently large?

Aside: Normal Distribution

A Normal distribution (gaussian) has the form:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Which has mean μ and standard deviation σ.

Its often written as $\mathcal{N}(\mu, \sigma)$

We write $v \sim \mathcal{N}(\mu, \sigma)$ to mean that v is a sample from this distribution.

Multivariate Regression

The loss for multivariate linear regression is

$$L(A) = \sum_{i=1}^{n} \left(x_i^T A^T - y_i^T\right) \underbrace{(Ax_i - y_i)}_{-\boldsymbol{n_i}}$$

which measures a squared error $\left(\hat{f}(x_i) - y_i\right)^2$ and has a probabilistic interpretation as the log likelihood of an additive gaussian noise process:

$$y_i = Ax_i + n_i$$

where $n_{ij} \sim \mathcal{N}(0, \sigma)$, which implies $P\left(n_{ij} = v\right) \propto \exp\left(-v^2/\left(2\sigma^2\right)\right)$.

So the log likelihood is $\propto -v^2$

Regularized Multivariate Regression

We assume the elements of A are also sampled from a prior distribution

$A_{ij} \sim \mathcal{N}(0, \sigma')$

This corresponds to adding a regularization term to the loss:

$$L(A) = \sum_{i=1}^{n} \left(x_i^T A^T - y_i^T\right)(Ax_i - y_i) + \lambda \sum_{i,j} A_{ij}^2$$

Computing the gradient as before we find that

$$\nabla_A L = 2A \sum_{i=1}^{n} x_i x_i^T - 2 \sum_{i=1}^{n} y_i x_i^T + 2\lambda A = 0$$

Regularized Multivariate Regression

Substituting: $\nabla_A L = 2AM_{xx} - 2M_{yx} + 2\lambda A = 0$

And the new solution is:

$$A = M_{yx}(M_{xx} + \lambda I)^{-1}$$

And notice that if the eigenvalues of M_{xx} were $\lambda_1, \dots, \lambda_m$, i.e. $M_{xx} v_i = \lambda_i v_i$, then the new eigenvalues of $(M_{xx} + \lambda I)$ are

$$\lambda_1 + \lambda, \dots, \lambda_m + \lambda$$

because $(M_{xx}+\lambda I)v_i = (\lambda_i + \lambda)v_i$

So $(M_{xx}+\lambda I)$ is positive definite for $\lambda > 0$, and the eigenvalues of its inverse are at most $\frac{1}{\lambda}$.

Regularized Multivariate Regression

The new solution

$$A = M_{yx}(M_{xx} + \lambda I)^{-1}$$

should be much more stable (lower variance).

However, the loss is no longer just the squared error:

$$L(A) = \sum_{i=1}^{n} (x_i^T A^T - y_i^T)(Ax_i - y_i) + \lambda \sum_{i,j} A_{ij}^2$$

so we don't expect the model to minimize squared error as strongly as before (higher bias).

Regularized Multivariate Regression

This is one type of regularization (L2 regularization) which is widely used with deep networks (its commonly implemented as "weight decay" in SGD optimizers).

$$L(A) = \sum_{i=1}^{n} \left(x_i^T A^T - y_i^T\right)(Ax_i - y_i) + \lambda \sum_{i,j} A_{ij}^2$$

Like other forms of regularization, L2 regularization allows you to trade off bias and variance.

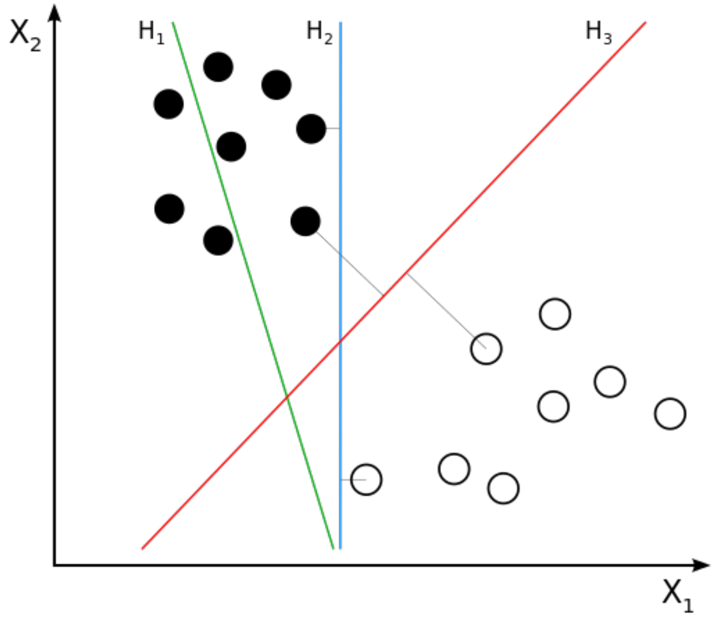
Strong regularization (large λ) ➔ lower variance and higher bias

Weak regularization (small λ) ➔ higher variance and lower bias

Support Vector Machines

Suppose we have some two-dimensional observations, i.e. pairs (x_1, x_2) that belong to two classes.

We can plot them in the plane as black and white points.



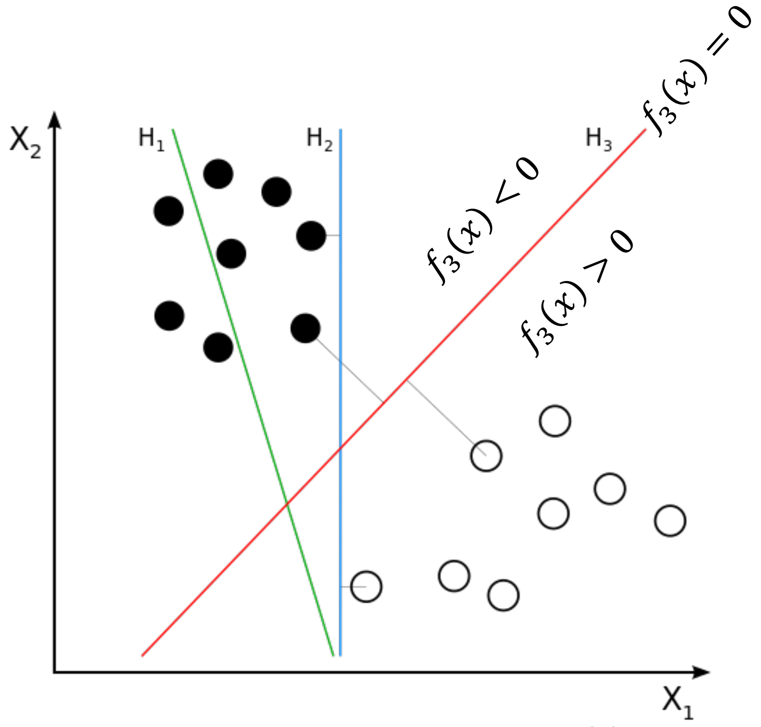
Support Vector Machines

We can construct a linear classifier with a weight vector $w = (w_0, w_1, w_2)$ and the decision function:

$$f(x) = w_0 + w_1 x_1 + w_2 x_2 > 0$$

The lines on the right show the decision boundaries for 3 different weight vectors.

i.e. they show where $f_j(x) = 0$ where $j \in \{1,2,3\}$ corresponding to H_1, H_2, H_3

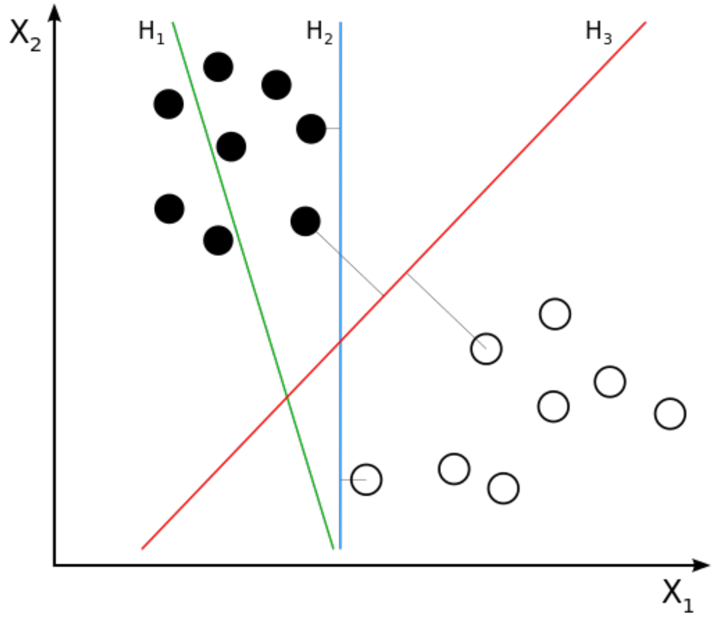


Support Vector Machines

H_1 does not correctly classify the data, while both H_2 and H_3 do.

But H_2 and H_3 do not define equally good classifiers.

Which (H_2 or H_3) boundary gives a better classifier and why?



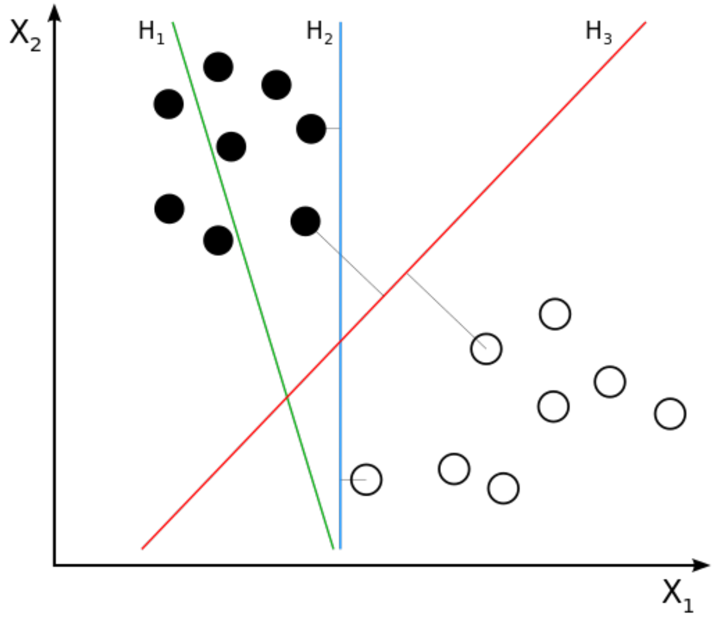
Support Vector Machines

H_1 does not correctly classify the data, while both H_2 and H_3 do.

But H_2 and H_3 do not define equally good classifiers.

Which (H_2 or H_3) boundary gives a better classifier and why?

H_3 has a larger ***margin*** (gray lines) and is more likely to correctly classify new data.

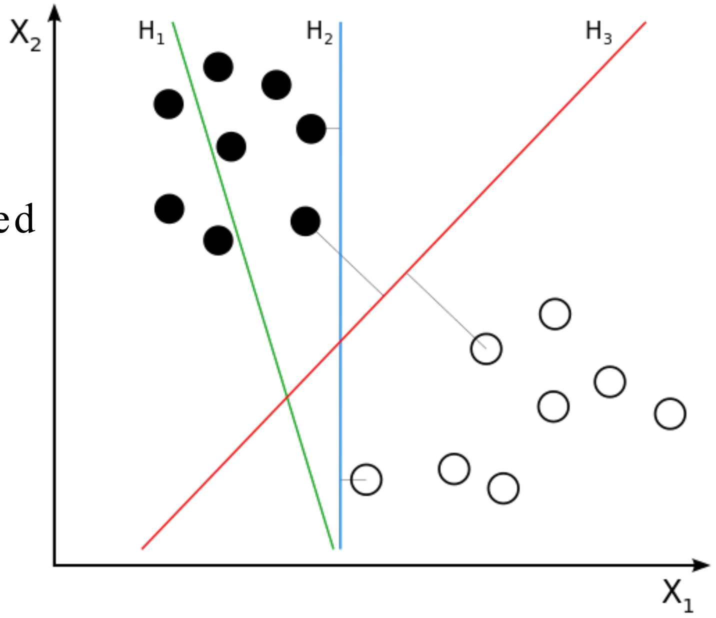


Support Vector Machines

H_3 has a larger ***margin*** (gray lines) and is more likely to correctly classify new data.

New data points have to be **at least as far away as the margin** from correctly classified points in order to be misclassified.

A large margin implies these points have to be far from current data (unlikely).

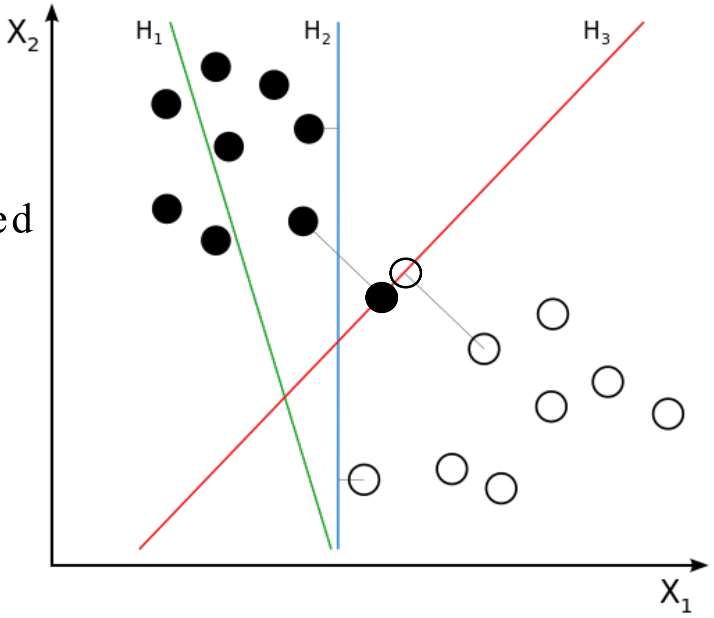


Support Vector Machines

H_3 has a larger ***margin*** (gray lines) and is more likely to correctly classify new data.

New data points have to be **at least as far away as the margin** from correctly classified points in order to be misclassified.

A large margin implies these points have to be far from current data (unlikely).

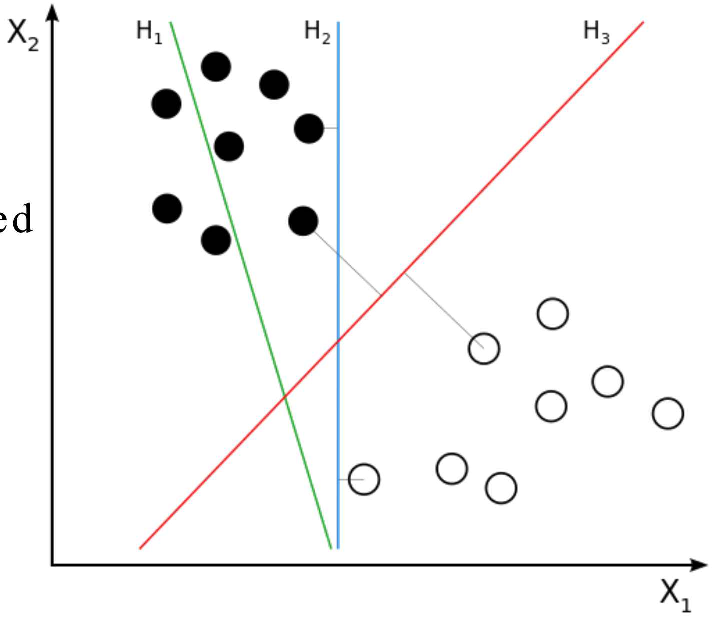


Support Vector Machines

H_3 has a larger ***margin*** (gray lines) and is more likely to correctly classify new data.

New data points have to be **at least as far away as the margin** from correctly classified points in order to be misclassified.

A small margin implies these points can be near current data (likely).

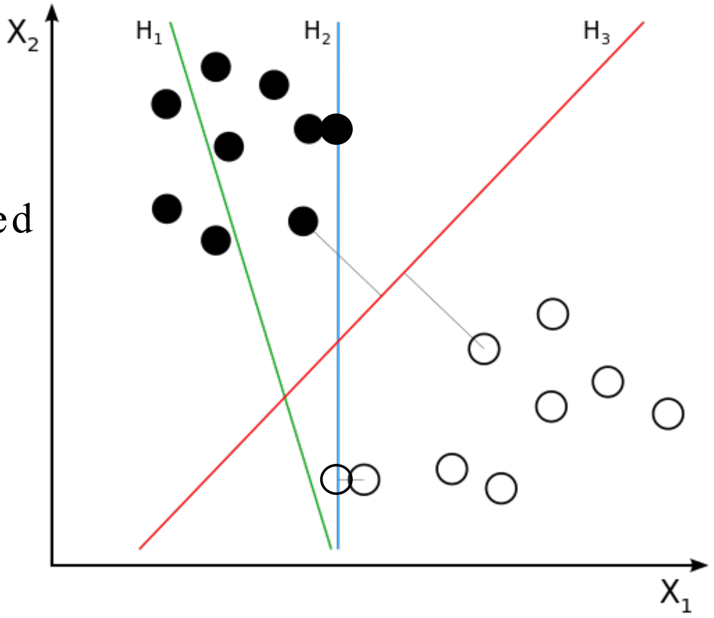


Support Vector Machines

H_3 has a larger ***margin*** (gray lines) and is more likely to correctly classify new data.

New data points have to be **at least as far away as the margin** from correctly classified points in order to be misclassified.

A small margin implies these points can be near current data (likely).



SVM Loss

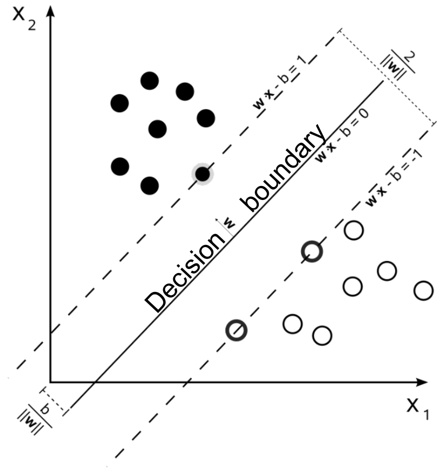
We would like to maximize the classifier's margin, which we do in several stages.

If the 2-norm (length) of the weight w vector $||w||_2 = 1$, then $f(x) = w^T x + b$ is a signed measure of the distance from x to the line $f(x) = 0$.

We can create a unit-length margin using the lines $f(x) = 1$ and $f(x) = -1$ (see figure on the right), and setting:

$f(x) \geq 1$ for $x \in C$

$f(x) \leq -1$ for $x \notin C$



x_2
x_1
$\mathbf{w}\cdot\mathbf{x} - b = 1$
$\mathbf{w}\cdot\mathbf{x} - b = 0$
$\mathbf{w}\cdot\mathbf{x} - b = -1$
Decision boundary
$\mathbf{w}$
$\frac{2}{||\mathbf{w}||}$
$\frac{b}{||\mathbf{w}||}$

Hinge Loss

Next we use a common labeling trick $y = \begin{cases} -1 & \text{if } \; x \notin \mathrm{C} \\ 1 & \text{if } \; \mathrm{x} \in \mathrm{C} \end{cases}$

And then the two inequalities
$f(x) \geq 1$ for $x \in C$
$f(x) \leq -1$ for $x \notin C$

Simplify to one:

$$yf(x) \geq 1$$

Then we want a loss that measures how much this constraint is ***violated*** . This value does it:

$$\max(0, 1 - yf(x))$$

This is called a ***hinge loss*** .

Hinge Loss

For a set of data points, the hinge loss is just the sum of per-point losses

$$l = \sum_{i=1}^{n} \max(0{,}1 - y_i f(x_i))$$

We assumed so far that $f(x) = w^T x + b$ with $||w||_2 = 1$.

As $||w||$ increases, the hinge loss decreases. Why?

Minimizing l creates a "pressure" on $||w||$ to increase. Instead of forcing $||w|| = 1$ (expensive), we add a loss term $\lambda||w||^2$ which tends to decrease $||w||$:

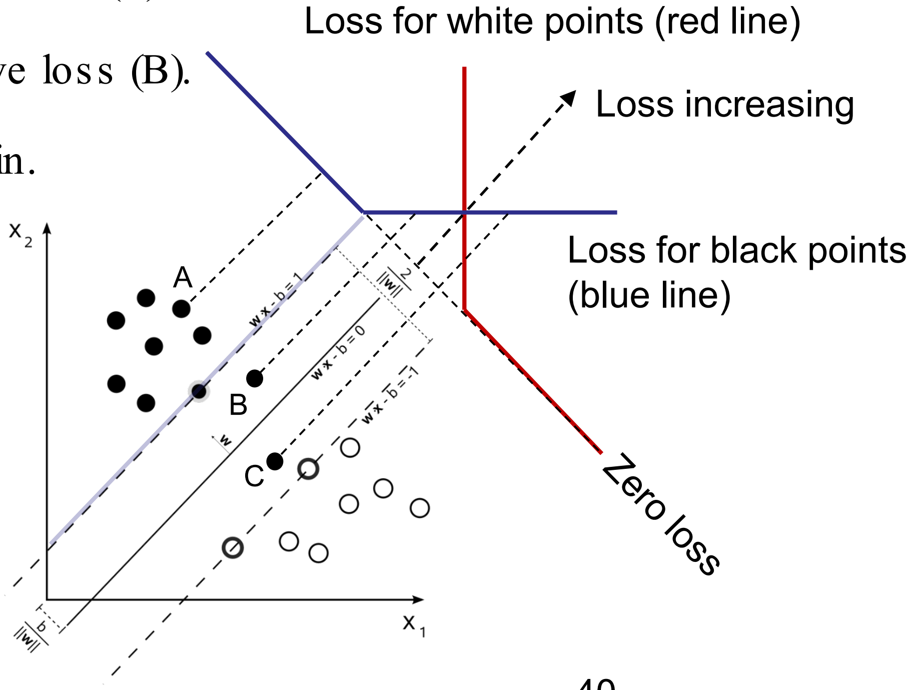
$$l = \sum_{i=1}^{n} \max(0{,}1 - y_i f(x_i)) + \lambda||w||^2$$

And λ can be adjusted to get $||w||$ close to 1. i.e. this classifier learns the margin as well as the boundary. This is a ***soft-margin*** SVM.

Hinge Loss

We can show the hinge losses for positive and negative instances on the figure:

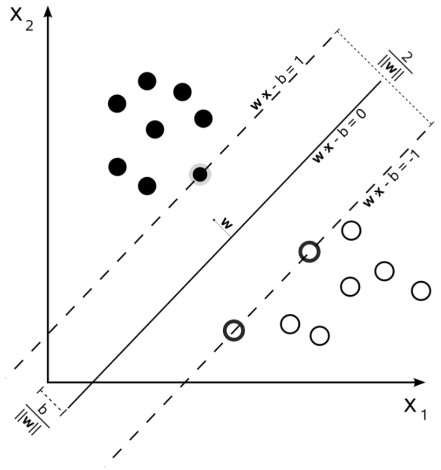
- Correct points outside the margin have 0 loss (A).
- Correct points within margin have positive loss (B).
- Loss grows with distance from the margin.
- Incorrect points have positive loss (C).



Loss for white points (red line)
Loss increasing
Loss for black points (blue line)
Zero loss
A
B
C
x_2
x_1
$\mathbf{w}\cdot\mathbf{x}-b=1$
$\mathbf{w}\cdot\mathbf{x}-b=0$
$\mathbf{w}\cdot\mathbf{x}-b=-1$
$\frac{2}{\|\mathbf{w}\|}$
$\frac{b}{\|\mathbf{w}\|}$
$\mathbf{w}$

Compare with Logistic Regression

How does the logistic regression (cross-entropy) loss vary for these points?



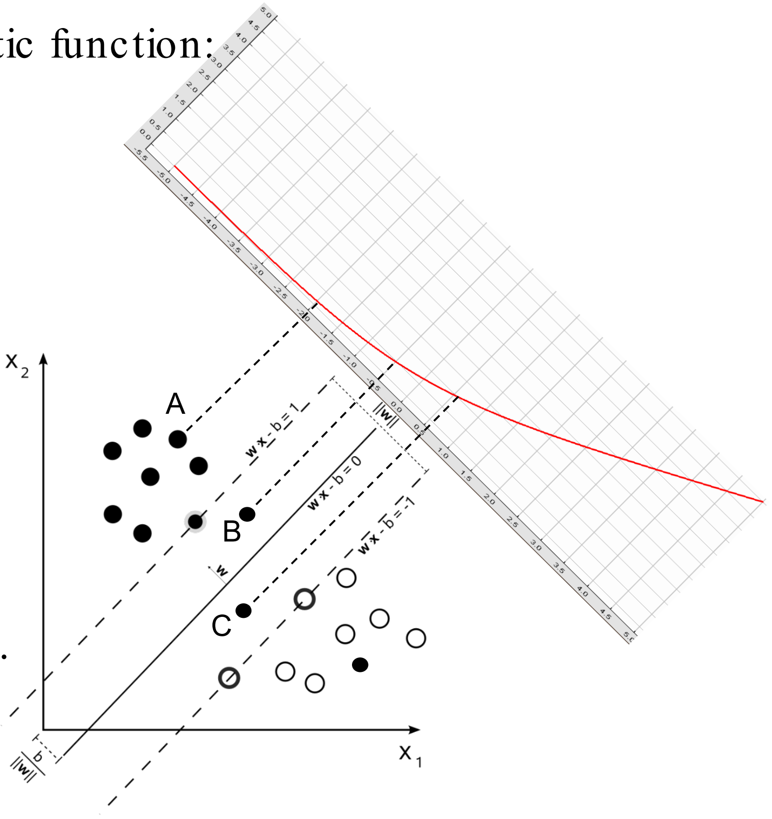
x_2
x_1
$\mathbf{w}\cdot\mathbf{x}-b=1$
$\mathbf{w}\cdot\mathbf{x}-b=0$
$\mathbf{w}\cdot\mathbf{x}-b=-1$
$\frac{2}{\|\mathbf{w}\|}$
$\frac{b}{\|\mathbf{w}\|}$
$\mathbf{w}$

Compare with Logistic Regression

Logistic regression loss is the (negative) log of the logistic function:

- $-\log(1/(1+\exp(-s)) = \log(1+\exp(-s))$
  where $s = w^T x$
- Small loss to correct points outside the margin (A)
- Larger loss to correct points in the margin (B)
- Still larger loss to incorrect points in the margin (C)

So logistic regression is a kind of ***soft-margin*** classifier.



Multi-Class Classification

One-versus-rest: For k classes, we construct k functions $f_i(x)$ for $i = 1, \ldots, k$, designed to estimate the probability that x is in class i.

We train $f_i(x)$ with labeled data: $y = 1$ if $x \in C$ and $y = 0$ if x is in any other class.

The predicted class label for x is

$$\underset{i}{\operatorname{argmax}} \quad f_i(x)$$

One-versus-one: For k classes we construct $\binom{k}{2}$ functions $f_{ij}(x)$ with $i < j \in \{1, \ldots, k\}$.

Each $f_{ij}(x)$ is a binary classifier for class i and class j, and is trained on those classes.

$f_{ij}(x) > 0$ is a vote for class i, while $f_{ij}(x) < 0$ is a vote for class j.

For a new instance x, tally the votes from all $\binom{k}{2}$ classifiers. Output the class with highest vote.

Multi-Class Logistic Regression

We consider the one-versus-rest design because its scalable.

We assume x is an m-dimensional vector. It may be binary or real-valued.

We would like $f_j(x)$ to estimate the probability that x is in class j.

Assume that we apply a $k \times m$ linear weight vector W to x, and let $s = Wx$.

We define:

$$f_j(x) = \frac{\exp(s_j)}{\exp(s_1) + \exp(s_2) + \cdots + \exp(s_k)}$$

Clearly

$$\sum_{j=1}^{k} f_j(x) = 1$$

So it acts as a probability

Multiclass Logistic Regression (Softmax)

The formula:

$$f_j(x) = \frac{\exp(s_j)}{\exp(s_1) + \exp(s_2) + \cdots + \exp(s_k)}$$

Is called a ***softmax*** of the vector $(s_1, \ldots, s_k)$.

The intuition for the name "softmax" is that if some s_j is somewhat larger than the others, its exponential will dominate the sum, and the output will be close to $(0, \ldots, 1, \ldots, 0)$ where the 1 is in the j^{th} position.

Multiclass Logistic Regression (Softmax)

The Softmax formula should generalize the 2-class formula from last time:

$$f_1(x) = \frac{\exp(s_1)}{\exp(s_1) + \exp(s_2)}$$

And the 2-class formula is

$$f(x) = \frac{1}{1 + \exp(-w^T x)}$$

To make this correspondence, $-w^T x$ should be:

A. $\exp(s_2 - s_1)$

B. $s_1 - s_2$

C. $s_2 - s_1$

D. $s_1 + s_2$

Skip this question

Oops

Substituting $-w^T x = \exp(s_2 - s_1)$ in the formula

$$f(x) = \frac{1}{1 + \exp(-w^T x)}$$

Would give $\exp(\exp(s_2 - s_1))$ in the denominator…

Try Again

Continue

Almost!

Substituting $-w^T x = s_1 - s_2$ would give

$$f(x) = \frac{1}{1 + \exp(s_1 - s_2)} = \frac{\exp(s_2)}{\exp(s_1) + \exp(s_2)}$$

Which is the multivariate formula for class 2 (not what we asked for).

Try Again

Continue

Right!

Substituting $-w^T x = s_2 - s_1$ would give

$$f(x) = \frac{1}{1 + \exp(s_2 - s_1)} = \frac{\exp(s_1)}{\exp(s_1) + \exp(s_2)}$$

Which is the multivariate formula for class 1.

Try Again

Continue

Oops!

Substituting $-w^T x = s_1 + s_2$ would give

$$f(x) = \frac{1}{1 + \exp(s_1 + s_2)} = \frac{\exp(-s_1)}{\exp(-s_1) + \exp(s_2)}$$

Which decreases as s_1 increases while the correct formula increases with s_1

Try Again

Continue

Cross-Entropy (Softmax) Loss

Cross-entropy loss is usually used with Softmax:

$$L = -\sum_{i=1}^{n} y_i^T \log f(x_i)$$

Where x_i is the i^{th} data sample and y_i represents the label of sample i.
Here y_i is a k-dimensional, one-hot vector having the form $(0, \ldots, 1, \ldots, 0)$ where the 1 is in the position of the correct label.

But that's not very compelling. Is there a more convincing argument for this particular choice of the Softmax function?

Relation to (multiclass) naïve Bayes

There is a very good reason: once again multiclass logistic regression (softmax of Wx) is ***the discriminative version of*** multiclass naïve bayes.

In particular, if x is a binary vector, there is a weight matrix W such that the softmax classifier implements the multiclass naïve bayes classifier (so it's the optimal classifier under the assumptions of naïve bayes, that features are independent given the class).

Proof: (check yourself)

Assume x has a constant coordinate x_0. Then define $W_{j,0}$

$$W_{j,0} = \log\big(P(X = 0|Y = j)P(Y = j)\big) = \log\big(P(X_1 = 0|Y = j)\cdots P(X_m = 0|Y = j)P(Y = j)\big)$$

And

$$W_{j,i} = \log\left(\frac{P(X_i = 1|Y = j)}{P(X_i = 0|Y = j)}\right) \quad \cdots$$

Weight Regularization

λ= regularization strength (hyperparameter)

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i^T \log f(x_i) + \boxed{\lambda R(W)}$$

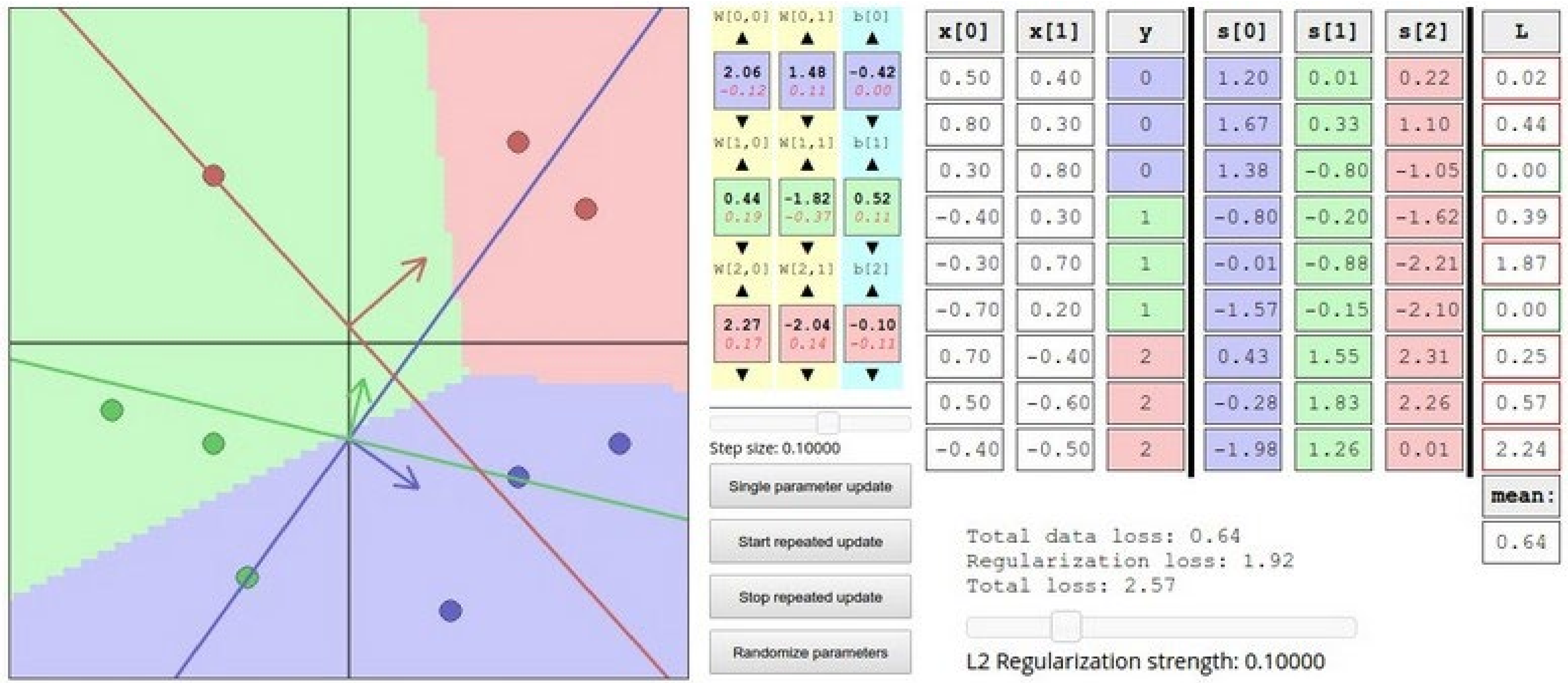
In common use:

L2 regularization	$R(W) = \sum_k \sum_l W_{k,l}^2$
L1 regularization	$R(W) = \sum_k \sum_l \lvert W_{k,l} \rvert$
Elastic net (L1 + L2)	$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + \lvert W_{k,l} \rvert$

Dropout (deep net specific)

Gradient noise, weight noise, MCMC simulation…

Interactive Web Demo time....



W[0,0]	W[0,1]	b[0]
2.06 -0.12	1.48 0.11	-0.42 0.00

W[1,0]	W[1,1]	b[1]
0.44 0.19	-1.82 -0.37	0.52 0.11

W[2,0]	W[2,1]	b[2]
2.27 0.17	-2.04 0.14	-0.10 -0.11

Step size: 0.10000

x[0]	x[1]	y	s[0]	s[1]	s[2]	L
0.50	0.40	0	1.20	0.01	0.22	0.02
0.80	0.30	0	1.67	0.33	1.10	0.44
0.30	0.80	0	1.38	-0.80	-1.05	0.00
-0.40	0.30	1	-0.80	-0.20	-1.62	0.39
-0.30	0.70	1	-0.01	-0.88	-2.21	1.87
-0.70	0.20	1	-1.57	-0.15	-2.10	0.00
0.70	-0.40	2	0.43	1.55	2.31	0.25
0.50	-0.60	2	-0.28	1.83	2.26	0.57
-0.40	-0.50	2	-1.98	1.26	0.01	2.24
						mean:
						0.64

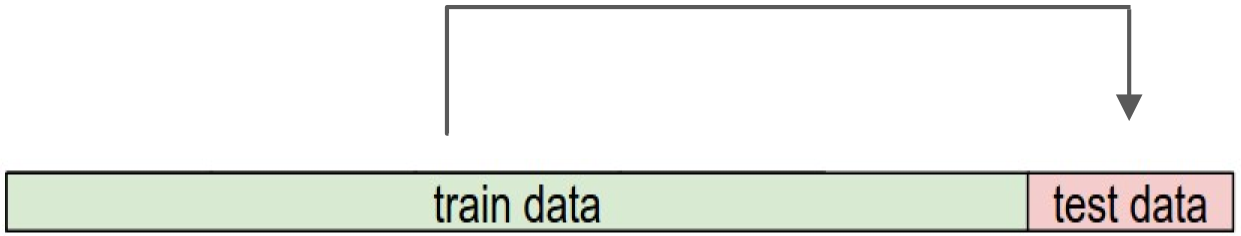
```
Total data loss: 0.64
Regularization loss: 1.92
Total loss: 2.57
```

L2 Regularization strength: 0.10000

http://vision.stanford.edu/teaching/cs231n/linear-classify-demo/

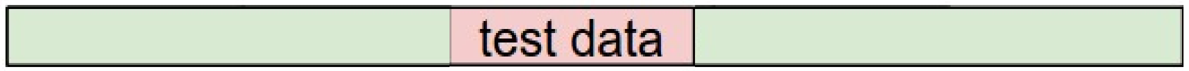
Slides based on cs231n by Fei-Fei Li & Andrej Karpathy & Justin Johnson

Measuring Performance



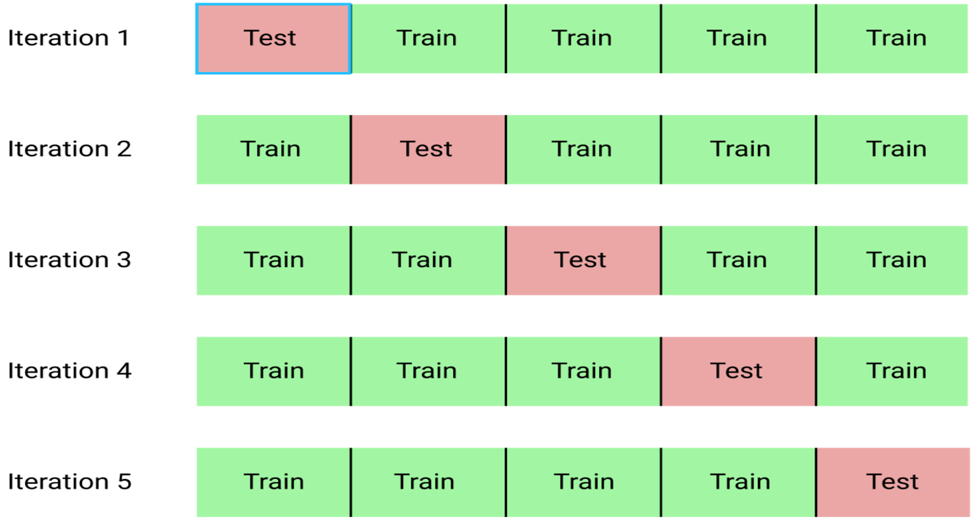
When we measure a classifiers performance, we want to use the most accurate model, so we want to use as much data as possible for training.
But we also want to use as much as possible for testing, in order to get the most accurate measurement.
It seems that we can only partition the data as above, but can we do better?

Cross Validation



Yes, we can make other partitionings of the data, like the one above.

If we keep the test samples disjoint, they will give mostly independent measurements. We do this by using a k-fold cross-validation design (k=5 here):

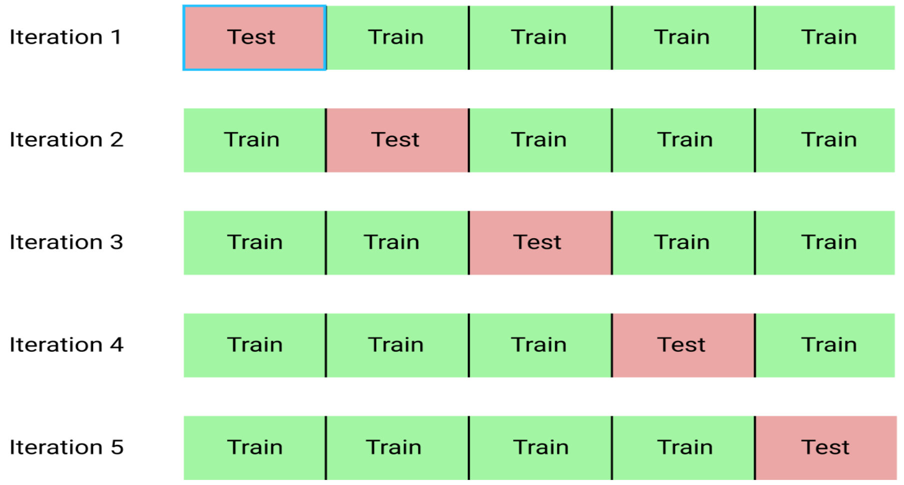


Cross Validation

For each iteration, we combine the 4 green folds into one training set, and train a model. We evaluate the model on the red fold for that iteration.

Finally we average performance over the 5 iterations.

This gives a more accurate performance estimate than a single run, although the improvement in stdev is less than $\sqrt{5}$



Summary

- We defined **bias** and **variance** and discussed the trade-off between them
- We discussed regularization to deal with over-fitting
- We defined a linear classifier’s **margin** and **hinge loss**.
- Maximizing the margin gives a Support Vector Machine (SVM) classifier.
- We then generalized logistic regression and cross-entropy loss to multiple classes, and derived a **softmax** classifier.
- We explored these methods with an **interactive visualization**.
- We touched on cross-validation as a way to more accurately test a model.