

Advanced course in machine learning
582744
Lecture 7

Arto Klami

Outline

Generative and discriminative models

Linear regression

Logistic regression

Sparse linear models

Supervised learning (from lecture 3)

The thing we all know:

- ▶ Input: \mathbf{x}_n, y_n
- ▶ Learning task: $f(\mathbf{x}_n) \approx y_n$
- ▶ Use: $f(\mathbf{x})$

Generative and discriminative models

Supervised tasks often solved with *discriminative models* that model the conditional density $p(y|\mathbf{x})$ of the labels given the covariates

No attention paid for $p(\mathbf{x})$, the distribution of the covariates

An alternative is to consider *generative classifiers* that construct a model for the joint density $p(y, \mathbf{x})$ and use it to infer the conditional density

$$p(y|\mathbf{x}) = \frac{p(y, \mathbf{x})}{p(\mathbf{x})}$$

Gaussian discriminant analysis

One practical generative classifier assumes $p(\mathbf{x}|y = c) = N(\mu_c, \Sigma_c)$

New samples classified using

$$\arg \min (\mathbf{x} - \mu_c)^T \Sigma_c^{-1} (\mathbf{x} - \mu_c),$$

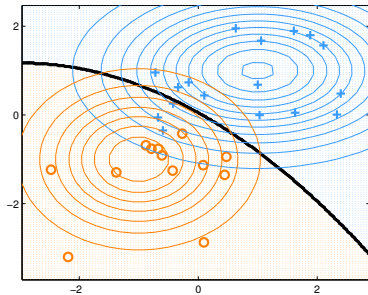
or by normalizing these to get the full posterior

If the covariances are shared ($\Sigma_c = \Sigma$), we get *Linear discriminant analysis* (LDA), since the discriminative boundary becomes linear

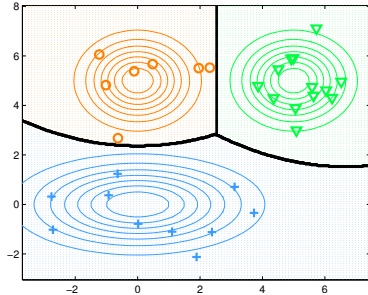
With general covariances the method is called *Quadratic discriminant analysis*

Generative classifiers

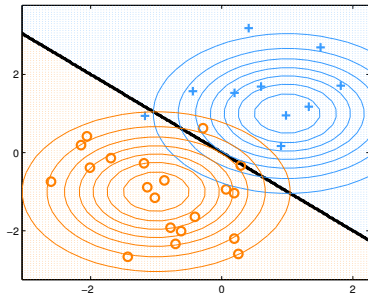
Parabolic Boundary



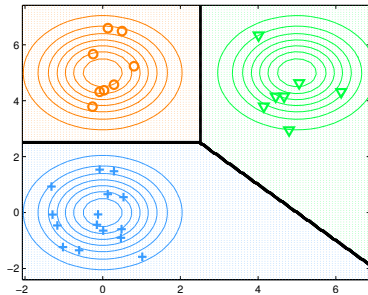
Some Linear, Some Quadratic



Linear Boundary



All Linear Boundaries



Generative vs discriminative (Section 8.6.1)

- ▶ Generative often easier to fit
- ▶ ...but we are fitting a more complex representation
- ▶ Missing inputs easier for generative models
- ▶ Unlabeled examples can help for generative models (semi-supervised learning)
- ▶ Discriminative models support arbitrary preprocessing steps for the inputs
- ▶ Generative models rely more on the assumptions of the model

The rest of the course is about discriminative models

Linear regression

Minimize the squared loss

$$L(\mathbf{w}) = \sum_n (y_n - \mathbf{w}^T \mathbf{x}_n)^2$$

Solution using pseudo-inverse

$$\mathbf{w} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y},$$

which is nothing but the solution for $\nabla L = 0$

Want bias term? Easiest to implement by adding $\mathbf{X}_{1n} = 1 \quad \forall n$ as the first element of the covariates

Linear regression

Probabilistic formulation

$$p(y_n | \mathbf{w}, \mathbf{x}_n, \sigma^2) = N(\mathbf{w}^T \mathbf{x}_n, \sigma^2)$$

gives the negative log-likelihood

$$L(\mathbf{w}, \sigma^2) = C + \log \sigma^2 + \frac{1}{2\sigma^2} \sum_n (y_n - \mathbf{w}^T \mathbf{x}_n)^2$$

Irrespective of σ^2 we need to minimize the same squared loss, but we also get an estimate for the residual noise

y_n can just as well be multivariate, making \mathbf{w} a matrix. We simply model each dimension independently

Linear regression vs PCA

The probabilistic models

$$\mathbf{y}_n \sim N(\mathbf{W}^T \mathbf{x}_n, \sigma^2 \mathbf{I}) \quad (\text{LR})$$

and

$$\mathbf{x}_n \sim N(\mathbf{W}^T \mathbf{z}_n, \sigma^2 \mathbf{I}) \quad (\text{PCA})$$

look awfully similar

One can think of PCA as linear regression where the covariates are unknown and need to be estimated as well

Regularized linear regression

Even though linear regression is extremely simple model, it can still overfit – we already saw this in exercise 2

The prior $\mathbf{w}_d \sim N(0, \tau^{-1})$ results in loss

$$L(\mathbf{w}, \sigma^2) = C + \log \sigma^2 + \frac{1}{2\sigma^2} \sum_n (y_n - \mathbf{w}^T \mathbf{x}_n)^2 + \frac{1}{2} \tau \mathbf{w}^T \mathbf{w},$$

and after some elementary calculus we get the MAP solution

$$\mathbf{w} = (\mathbf{X}\mathbf{X}^T + \alpha \mathbf{I})^{-1} \mathbf{X}\mathbf{y},$$

where α is $\sigma^2 \tau$ (from the perspective of \mathbf{w} some constant)

This is called *ridge regression*, which has the effect of pulling the elements of \mathbf{w} towards zero. Cross-validation can be used for choosing α .

Robust linear regression

What if we have outliers?

Student's t-distribution has heavier tails than normal distribution, and hence

$$y_n \sim t_\nu(\mathbf{w}^T \mathbf{x}_n, \sigma^2)$$

gives a more robust linear regression model

Laplace distribution $\log p \propto |y_n \mathbf{w}^T \mathbf{x}_n|$ would also work

No closed-form solution, so optimization is harder

Logistic regression

Generalized linear models (Section 9; not covered on the course) are linear models that use other exponential family densities in place of the Gaussian likelihood

Logistic regression is one important special case:

$$p(y_n|\mathbf{w}, \mathbf{x}_n) = \text{Ber}(y|\text{sigm}(\mathbf{w}^T \mathbf{x}_n))$$

We already saw the negative log-likelihood in exercise 2.1, and even derived the gradient for minimizing it

In matrix-form the prettiest equations are probably

$$\mathbf{g} = \mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y})$$

$$\mathbf{H} = \mathbf{X}^T \mathbf{S} \mathbf{X},$$

where $\boldsymbol{\mu}_n = \text{sigm}(\mathbf{w}^T \mathbf{x}_n)$ and $\mathbf{S} = \text{diag}(\boldsymbol{\mu}_n(1 - \boldsymbol{\mu}_n))$

Logistic regression

Newton's method solves the problem well, and can be re-interpreted as *iteratively reweighted least squares* (IRLS) because it iteratively solves optimization problems of the form

$$L(\mathbf{w}) = \sum_n s_n (z_n - \mathbf{w}^T \mathbf{x}_n)^2$$

where s_n are weights and z_n are pseudo-targets (see Section 8.3.4)

Regularized version with the prior $\mathbf{w}_{nd} \sim N(0, \alpha^{-1})$ simply adds $\alpha \mathbf{w}$ for the gradient and $\alpha \mathbf{I}$ for the Hessian

The model can also be extended for multiple outputs, as *multinomial logistic regression*, using the likelihood

$$p(y_n = c | \mathbf{W}, \mathbf{x}_n) = \frac{e^{\mathbf{w}_c^T \mathbf{x}_n}}{\sum_j e^{\mathbf{w}_j^T \mathbf{x}_n}}$$

The updates are given in Section 8.3.7; these are a bit messy and the Hessian is huge

Sparsity

Linear models overfit when D is large compared to N ; in extreme cases $D \gg N$, for example $D = 10,000$ genes used as covariates for $N = 100$ patients

...or because we expanded the feature space to get a non-linear variant

One intuitively effective way of regularizing the model for such cases is *feature selection*, which means using only a subset of the original D dimensions in the model

Plenty of techniques for selecting them in advance, but here we consider approaches that directly learn *sparse* solutions, implicitly selecting the features

Direct sparsity

Probabilistic formulation: Introduce new latent variables

$\gamma_d \sim \text{Ber}(\theta)$ that tell whether a feature is being used or not, and infer $p(\gamma|D) \propto p(D|\gamma)p(\gamma)$

The prior is $p(\gamma) = \theta^{\|\gamma\|_0} (1 - \theta)^{D - \|\gamma\|_0}$, where $\|\gamma\|_0$ is the l_0 -norm that counts how many elements differ from zero

By further specifying that $\mathbf{w}_d = 0$ if $\gamma_d = 0$ and $\mathbf{w}_d \sim N(0, \alpha)$ if $\gamma_d = 1$, we can write a full probabilistic model

In the end it gives a regularization term $\lambda \|\gamma\|_0$

Direct sparsity

Optimization for losses $L(\mathbf{w}) + \lambda \|\mathbf{w}\|_0$ is difficult because the regularizer is not smooth

- ▶ Greedy choice: Start with $\mathbf{w} = 0$ and always add the best dimension
- ▶ Backwards selection: Start with $\gamma_d = 1 \quad \forall d$ and remove features greedily
- ▶ ...and a wide range of more advanced algorithms, including EM for a slightly modified prior

In practice we usually do not directly learn such a sparse model but try to find sparse models by other means

Sparsity

The l_2 norm gave ridge regression that simply pulls the weights towards zero, whereas l_0 counted the number of non-zero elements and hence directly enforces sparsity

What if we tried regularizing with l_p for some $0 \leq p \leq 2$, such as l_1 ?

Corresponds to the Laplace prior $e^{-\lambda \|w_d\|}$

Sparsity via regularization

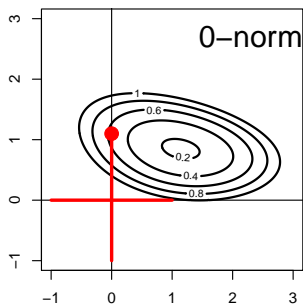
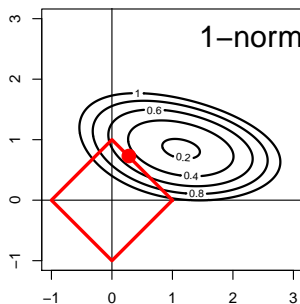
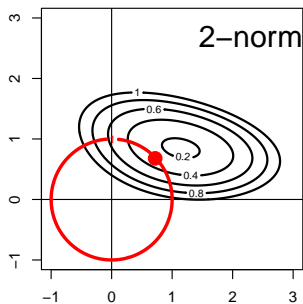
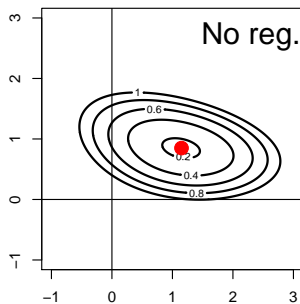
Side-remark:

Minimizing $L(\mathbf{w}) + \lambda R(\mathbf{w})$ is equivalent to minimizing $L(\mathbf{w})$ such that $R(\mathbf{w}) \leq B$ for some B , where smaller B correspond to larger λ

Might help geometric intuition: Regularized solution has to stay within a “ball” of given norm

...but the relationship between B and λ above depends on the data

Sparsity – geometric motivation



Sparse linear regression

Regression with l_1 regularization is called *lasso* for “least absolute shrinkage and selection operator”

$$L(\mathbf{w}) = \sum_n (y_n - \mathbf{w}^T \mathbf{x}_n)^2 + \lambda \|\mathbf{w}\|_1$$

The partial derivative of the regularizer is not defined at $\mathbf{w}_d = 0$, but we can use *subgradient* which is any slope that touches the loss function at such discontinuity and is below the curve

We find the minimum by following the subgradients, and for the solution the zero vector belongs to the family of subgradients

Coordinate descent for Lasso

The derivative of the unregularized loss wrt to w_d is given by

$$\begin{aligned}\frac{\partial L(\mathbf{w})}{\partial w_d} &= -2 \sum_n x_{nd} (y_n - \mathbf{w}^T \mathbf{x}_n) \\ &= -2 \sum_n \left(x_{nd} (y_n - \mathbf{w}_{-d}^T \mathbf{x}_{n,-d}) + 2x_{nd}^2 w_d \right) \\ &= a_d w_d - c_d,\end{aligned}$$

where $a_d = 2 \sum_n \mathbf{x}_{nd}^2$ and $c_d = 2 \sum_n \mathbf{x}_{nd} (y_n - \mathbf{w}_{-d}^T \mathbf{x}_{n,-d})$

The subgradient of the regularizer is

$$\frac{\partial \|\mathbf{w}\|_1}{w_d} = \begin{cases} -\lambda & \text{when } w_d < 0 \\ [-\lambda, \lambda] & \text{when } w_d = 0 \\ \lambda & \text{when } w_d > 0 \end{cases}$$

Lasso (Section 13.3.2)

The full subgradient is then given by

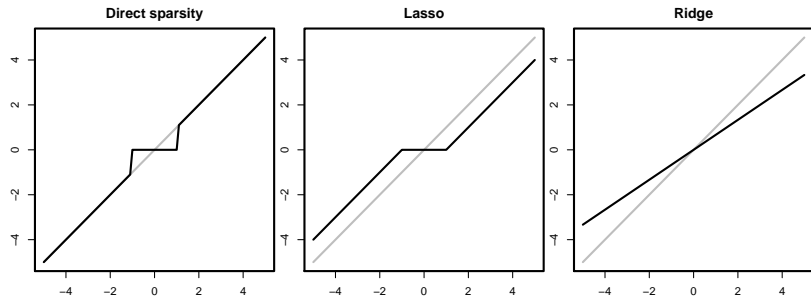
$$\nabla L(\mathbf{w}) = \begin{cases} a_d w_d - c_d - \lambda & \text{if } w_d < 0 \\ [-c_d - \lambda, -c_d + \lambda] & \text{if } w_d = 0 \\ a_d w_d - c_d + \lambda & \text{if } w_d > 0, \end{cases}$$

Solution identified when zero belongs to the sub-gradient

$$\begin{aligned} w_d &= (c_d + \lambda)/a_d && \text{if } c_d < -\lambda \\ w_d &= 0 && \text{if } c_d \in [-\lambda, \lambda] \\ w_d &= (c_d - \lambda)/a_d && \text{if } c_d > \lambda \end{aligned}$$

Pulls all weights towards zero by a constant amount, setting them exactly to zero if close enough

l_0 vs l_1 vs l_2



Both l_0 and l_1 put some weights exactly to zero, but l_2 does not

Lasso

An algorithm called LARS (“least angle regression and shrinkage”) can compute the solution for all values of λ in roughly the same time as it takes to solve the problem for one value

Based on the observation that the set of non-zero values changes only for certain critical values of λ and otherwise the weights change linearly

Other sparsity priors

- ▶ Group-lasso: l_1 -norm for groups of variables
- ▶ Elastic net: Sum of l_2 and l_1
- ▶ Bridge: $\sum_d \|w_d\|^b$ for $b \geq 0$ (note that $b = 1$ is the smallest one that is convex)
- ▶ Automatic relevance determination (ARD): $w_d \sim N(0, \alpha)$, with some flat prior for α (often used with PCA)

