CS489: Machine Learning

Michael Noukhovitch

Winter 2017,

Notes written from Pascal Poupart's lectures.

Contents

1 Introduction

machine learning giving computers ability to learn without being explicitly programmed

A machine learns from experience E wrt to some class of tasks T and performance measure P if its performance in task T, as measured by P, improves with E

1.1 Supervised Learning

Definition. given a training set of examples $(x, f(x))$, return a hypothesis h that approximates h

Two types:

classification where output space consists of *categorical* values

regression where output space consists of *numerical* values

1.1.1 Hypothesis Space

hypothesis space set of all hypotheses H that the learner may consider

consistent if hypothesis h agrees with f on all examples

realizable if the hypothesis space constains the consistent function

our objective can be restated as a search problem to find the hypothesis h in hypothesis space H that minimizes some objective

1.2 Unsupervised Learning

2 Nearest Neighbour

2.1 Basic NN

nearest neighbours label any example with the label of its nearest neighbours

classification: $h(x) = y_{x*}$
where $y_{x*} = \text{argmin}_{x'} d(x, x')$ is the label associated with the nearest neighbour

2.2 KNN

k-nearest neighbours assign the most frequent label among k nearest neighbours

let $knn(x)$ be the k nearest neighbours
then $y_x = \text{mode}(y_{x'} | x' \in knn(x))$

overfitting a hypothesis h with training accuracy higher than its own testing accuracy $\max(0, \text{trainAccuracy}(h) - \text{testAccuracy}(h))$

- classifier too expressive
- noisy data

- lack of data

underfitting a hypothesis h with training accuracy lower than testing accuracy of some other hypothesis h', $\max(0, \max_{h'} \text{trainAccuracy}(h) - \text{testAccuracy}(h'))$

- classifier not expressive enough

k-fold cross validation split data in k equal subsets, run k experiments testing on one subset, and training on all the others. Report average accuracy

weighted knn weight each neighbour by distance

knn regression y_x is a real value, $y_x \leftarrow average(y_{x'}|x' \in knn(x))$

3 Linear Regression

3.1 Least Squares

find linear hypothesis h: $t = w^T\bar{x}$, find w to minimize euclidean L2 loss

$$w* = argmin_w \frac{1}{2}\sum_{n=1}^{N}(t_n - w^T\bar{x}_n)^2$$

where $\bar{x} = \begin{pmatrix}1\\x\end{pmatrix}$ and we can solve with $w = A^{-1}b$ or $Aw = b$ which can be solved as a linear system

3.1.1 Regularization

Least squares can be unstable, overfit, so change optimization

$$w* = argmin_w \frac{1}{2}\sum_{n=1}^{N}(t_n - w^T\bar{x}_n)^2 + \frac{\lambda}{2}\|w\|_2^2$$

or $(\lambda I + A)w = b$

3.2 Maximum Likelihood

derive the same thing but from a different perspective: assume $y = w^T\bar{x}+$ gaussian noise so

$$\begin{aligned} Pr(y|\bar{X}, w, \sigma) &= N(y|w^T\bar{X}, \sigma^2) \\ w* &= argmax_w Pr(y|\bar{X}, w, \sigma) \\ &... \\ &= argmin_w \sum_n (y_n - w^T\bar{x}_n)^2 \end{aligned}$$

which is the same as least squares

3.3 Maximum A Posteriori

find $w*$ with highest posterior probability, knowing that prior $P(w) = N(0, \Sigma)$

$$Pr(w|X, y) \propto Pr(w)Pr(y|X, w)$$

therefore for optimization:

$$\begin{aligned} w* &= argmax_w Pr(w|\bar{X}, y) \\ &... \\ &= argmin_w \sum_n (y_n - w^T \bar{x}_n)^2 + w^T \Sigma^{-1} w \end{aligned}$$

let $\Sigma^{-1} = \lambda I$

$$= argmin_w \sum_n (y_n - w^T \bar{x}_n)^2 + \lambda \|w\|_2^2$$

and we arrive at least squares with regularization

3.4 Expected Squared Loss

$$\begin{aligned} E[loss] &= \int_{x,y} Pr(x, y)(y - w^T \bar{x})^2 dxdy \\ &= \int_{x,y} Pr(x, y)(y - f(x))^2 + \int_x Pr(x)(f(x) - w^T \bar{x})^2 dx \\ &= \text{noise (constant)} + \text{error (relative to } w) \end{aligned}$$

lets consider the expected error wrt our dataset S

$$\begin{aligned} E[error] &= E_S[(f(x) - \bar{w_S^T})^2] \\ &= (f(x) - E_S[w_S^T \bar{x}])^2 + E_S[(E_S[w_S^T \bar{x}] - w_S^T \bar{x})^2] \\ &= \text{bias}^2 + \text{variance} \end{aligned}$$

therefore putting it together

$$E[loss] = \text{bias}^2 + \text{variance} + \text{noise}$$

3.5 Bayesian Linear Regression

instead of using $w*$, compute weighted avg prediction using $Pr(w|\bar{X}, y)$

$$Pr(w|\bar{X}, y) = N(\bar{w}, A^{-1})$$

where $w = \sigma^{-2} A^{-1} \bar{X}^T y$

$A = \sigma^{-2} \bar{X}^T \bar{X} + \Sigma^{-1}$

let x_* be the input for which we predict y_*

$$\begin{aligned} Pr(y_*|\bar{x}_*, \bar{X}, y) &= \int_w Pr(y_*|\bar{x}_*, w) Pr(w|\bar{X}, y) dw \\ &\dots \\ &= N(\bar{x}_*^T A^{-1} \bar{X}^T y, \bar{x}_*^T A^{-1} \bar{x}_*) \end{aligned}$$

4 Statistical Learning

4.1 Introduction

probability distribution a specific probability for each event in our sample space

joint distribution spec of probabilities for all combinations of events $Pr(A \wedge B)$

conditional probabilties $Pr(A|B) = Pr(A \wedge B)/Pr(B)$

4.2 Bayes Rules

$$Pr(B|A) = \frac{Pr(A|B)Pr(B)}{Pr(A)}$$

posterior P(B|A)

likelihood P(A|B)

prior P(B)

normalizing P(A)

evidence A

4.3 Bayesian Learning

computing the posterior of hypothesis given evidence using Bayes' theorem:

$$Pr(H|e) = kPr(e|H)Pr(H)$$

properties:

+ optimal (given prior)
+ no overfitting (all hypotheses considered)
- intractable if hypothesis space is large

4.4 Approximate Bayesian Learning

Maximum A Posteriori make prediction based on most probable hypothesis (vs basing on all hypotheses weighted by probability)

$$h_{map} = argmax_{h_i} Pr(h_i|e)$$

+ controlled overfitting
+ converges as data increases
- less accurate than Bayesian prediction
- maybe be intractable!

Maximum Likelihood simplify MAP by assuming uniform prior $Pr(h_i) = Pr(h_j) \forall i, j$

$$h_{ml} = argmax_{h_i} Pr(e|h_i)$$

+ still converges
- least accurate because ignore prior info
- overfits

also, can be easier than MAP: $h_{ml} = argmax_h \sum_n \log Pr(e_n|h)$

5 Mixture of Gaussians

5.1 Introduction

Assume:

- each prior is frequency: $Pr(C = c_k) = \pi_k$
- $Pr(x|C)$ is gaussian
- covariance matrix Σ is used for each class $Pr(x|c_k) \propto e^{-\frac{1}{2}(x-\mu_k)^T \Sigma^(-1)(x-\mu_k)}$

Where:

$$\pi = \frac{\sum_n y_n}{N} \qquad \text{average } y$$

$$\mu_k = \frac{\sum_{n \in c_k} x_n}{N_k} \qquad \text{mean of class } k$$

$$\Sigma = \frac{N_1}{N} S_1 + \frac{N_2}{N} S_2 \ldots \qquad \text{covariance}$$

$$S_k = \frac{1}{N_k} \sum_{n \in c_k} (x_n - \mu_k)(x_n - \mu_k)^T \qquad \text{weighted variance}$$

5.2 Two Class

Then if there are two classes c_k and c_j,

$$\begin{aligned} Pr(c_k|x) &= \frac{1}{1+e^{-w^T x+w_0)}} \\ &= \sigma(w^T x + w_0) \end{aligned}$$

choose the best class as the one with probability > 0.5, so class boundary is at

$$\begin{aligned} \sigma(w_k^T x + w_0) &= 0.5 \\ w_k^T \bar{x} &= 0 \text{ is a linear separator} \end{aligned}$$

5.3 Multi-class

Normalize using softmax:

$$\begin{aligned} Pr(c_k|x) &= \frac{Pr(c_k)Pr(x|c_k)}{\sum_j Pr(c_j)Pr(x|c_j)} \\ &= \frac{e^{w_k^T \bar{x}}}{\sum_j e^{w_j^T \bar{x}}} \end{aligned}$$

6 Logistic Regression

6.1 Introduction

MoG is restrictive, assumes everything is a gaussian. Generalize to exponential family:

$$Pr(x|\Theta_k) = \exp(\Theta_k^T T(x) - A(\Theta_k) + B(x))$$

where Θ_k : parameters of class k

$T(x), A(\Theta_k), B(x)$: arbitrary functions

and the posterior $Pr(c_k|x) = \sigma(w^T x + w_0)$ which we will learn directly by maximum likelihood, in general it is

- **logistic sigmoid** for binary
- **softmax** for multiclass

6.2 Logisitc Regression Classification

For some dataset (X, y) and for two classes $y \in 0, 1$:

$$w^* = argmax_w \prod_n \sigma(w^T \bar{x}_n)^{y_n} (1 - \sigma(w^T \bar{x}_n))^{1-y_n}$$

so our objective is

$$L(w) = -\sum_n y_n \ln \sigma(w^T \bar{x}_n) + (1 - y_n) \ln(1 - \sigma(w^T \bar{x}_n))$$

finding the min by setting derivative to 0

$$\begin{aligned} \frac{dL}{dw} &= 0 \\ 0 &= \sum_n [\sigma(w^T \bar{x}_n) - y_n] \bar{x}_n \end{aligned}$$

and since we can't isolate w we use **Newton's Method** to iteratively solve:

$$w \to w - H^{-1} \nabla L(w)$$

where $H = \bar{X} R \bar{X}^T$ is the hessian

$$R = \begin{bmatrix} \sigma_1(1-\sigma_1) & & \\ & ... & \\ & & \sigma_N(1-\sigma_N) \end{bmatrix}$$

$$\sigma_k = \sigma(w^T \bar{x}_k)$$

6.3 Regularization

To ensure that we can inverse H (so it isn't singular), add λ:

$$H = \bar{X} R \bar{X}^T + \lambda I$$

6.4 Non-linear Regression

Non-linear regression using the same algorithm, map inputs to a different space! Use non-linear basis functions ϕ_i, so for

$\phi_0(x) = 1$
$\phi_1(x) = x$
$\phi_2(x) = x^2$

the hypothesis space is $H = \{x \leftarrow w_0\phi_0(x) + w_1\phi_1(x) + w_2\phi_2(x) | w_i \in \Re\}$

common basis functions:

- polynomial $\phi_j(x) = x^j$
- gaussian $\phi_j(x) = e^{-\frac{(x-\mu_j)^2}{2s^2}}$
- sigmoid $\phi_j(x) = \sigma(\frac{x-\mu_j}{s})$
- fourier, wavelets ...

7 Perceptron

7.1 Computer vs Brain

Computer:

- bunch of gates
- electrical signals by gates
- sequential and parallel
- fragile

Brain

- network of neurons
- nerve signal propogate
- parallel
- robust (neurons die)

ANN Unit consists of weights w and activation function h, so that output $y_j = h(W_j \bar{x})$. Structure is either **feed-forward** or **recurrent**

7.2 Perceptron Learning

Learning is done separately for each unit j:

for all pairs (x, y) **do**
  if output is correct **then**
    Do nothing
  else if output = 0, label = 1 **then**
    $\forall_i W_{ji} \to W_{ji} + x_i$
  else if output = 1, label = 0 **then**
    $\forall_i W_{ji} \to W_{ji} - x_i$
  end if
end for

7.3 Alternative Learning

let M be the set of misclassified examples (where $y_n w^T \bar{x}_n < 0$) then find w to minimize number of misclassifications:

$$E(w) = -\sum_{(x_n, y_n) \in M} y_n w^T \bar{x}_n$$

Use **gradient descent**

$$w \leftarrow w - \eta \nabla E$$

where η is the learning rate

If we adjust w one example at a time, we use **sequential gradient descent** which is equivalent to threshold perceptron learning when $\eta = 1$

7.4 Linear Separability

Threshold perceptron converges iff the data is linearly separable

linear separator $w^T \bar{x}$, since it is linear in w

7.5 Other Networks

Sigmoid Perceptron "soft" linear separators (same H as linear regression)

$$E(w) = \frac{1}{2} \sum_n (y_n - \sigma(w^T \bar{x}_n))^2$$

for all (x_n, y_n) **do**
$E_n \leftarrow y_n - \sigma(w^T \bar{x}_n)$
$w \leftarrow w + \eta E_n \sigma(w^T \bar{x}_n)(1 - \sigma(w^T \bar{x}_n))\bar{x}_n$
end for

8 Multilayer Neural Networks

8.1 Introduction

Previously, our basis functions were fixed, but we can remove that restriction by learning non-linear basis functions.

$$\text{hidden unit } z_j = h_1(w_j^{(1)} \bar{x})$$
$$\text{output unit } y_k = h_1(w_j^{(2)} \bar{x})$$
$$\text{both units } y_k = h_2(\sum_j w_{kj}^{(2)} h_1(\sum_i w_{ji}^{(1)} x_i))$$

if we consider our hidden input to be a basis function, then this is equivalent to a linear regression and a learned basis function, e.g.

- non-linear regression: h_1 is a non-linear function, h_2 is identity
- non-linear classification: h_1 is a non-linear function, h_2 is sigmoid

8.2 Backpropogation

Error function:

$$E(w) = \frac{1}{2} \sum_n \|f(x_n, W) - y_n\|_2^2$$
$$\text{where } f(x, W) = \sum_j w_{kj}^{(2)} \sigma(\sum_i w_{ji}^{(1)} x_i)$$

so our update rule for gradient descent is

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\delta E_n}{\delta w_{ji}}$$
$$\text{where } \frac{\delta E_n}{\delta w_{ji}} = \delta_j z_i$$

Do gradient update in two phases:

1. **forward**: compute output z_j of each unit j

$$z_j = h(\sum_i w_{ji} z_i)$$

2. **backward**: compute delta δ_j at each unit j

$$d_j = \begin{cases} h'(a_j)(z_i - y_i), & \text{if } j \text{ is output} \\ h'(a_j) \sum_k w_{kj} \delta_k & \text{if } j \text{ is a hidden unit before } k \end{cases}$$

Analysis:

- fast computation
- slow convergence, may get trapped in local optima
- prone to overfitting, solve with
  - early stopping
  - regularization

9 Kernel Methods

9.1 Introduction

Data may not be linearly separable, so map into a high-dimensional space where it is! This is computationally difficult though, so instead calculate a similarity measure (dot product) in the high dimensional space and use algorithms that only need that measure.

kernel methods use large set of fixed non-linear basis functions, with a "dual trick" to make complexity depend on amount of data instead of number of basis functions

kernel function $k(x, x') = \phi(x)^T \phi(x')$ for some basis function $\phi(x)$

linear regression objective, setting derivative to 0 gives us

$$w = -\frac{1}{\lambda} \sum_n (w^T \phi(x_n) - y_n) \phi(x_n)$$

so w is a linear combination of inputs in feature space

$$\begin{aligned} &= \{\phi(x_n) | 1 \leq n \leq N\} \\ \text{substitute } w &= \phi a \\ \text{where } \phi &= [\phi(x_1) \dots \phi(x_n)] \\ a &= [a_1, \dots a_n]^T \end{aligned}$$

now let $K = \Phi^T \Phi$, therefore our prediction

$$\begin{aligned} y_* &= \phi(x_*)^T \Phi a \\ &= k(x_*, X)(K + \lambda I)^{-1} y \end{aligned}$$

For this we need to just find dual solution a instead of w

- now depends on # of data instead of # of basis function
- can use many more basis functions
- don't actually need Φ, just need a semi-definite kernel $K, \exists \Phi \mid K = \Phi^T \Phi$ or all eigenvalues ≥ 0

9.2 Common Kernels

Common kernels:

- polynomial $k(x, x') = (x^T x' + c)^M, c \geq 0$
- gaussian $k(x, x') = \exp(-\frac{\|x-x'\|^2}{2\sigma^2})$

Also construct more kernels using rules. Let $k_1(x, x')$ and $k_2(x, x')$ be valid kernels, and $x = \begin{pmatrix} x_a \\ x_b \end{pmatrix}$ then it is also valid $k(x, x') =$

- $ck_1(x, x') \ \forall c > 0$
- $f(x)k_1(x, x')f(x') \ \forall f$
- $q(k_1(x, x'))$ where q is a polynomial with coeffs ≥ 0
- $\exp k_1(x, x')f(x')$
- $k_1(x, x') + k_2(x, x')$
- $k_1(x, x')k_2(x, x')$
- $k_3(\phi(x), \phi(x'))$
- $x^T A x'$ where A is symmetric positive semi-definite
- $k_a(x_a, x'_a) + k_b(x_b, x'_b)$
- $k_a(x_a, x'_a)k_b(x_b, x'_b)$

Kernels can also be defined wrt to sets, strings, graphs. E.g. $k(d_1, d_2) =$ similarity between two documents

9.3 Example

Show that $k(x, z) = (x^T z)^2$ is a valid kernel by finding ϕ

$$\begin{aligned} k(x, z) &= (x^T z)^2 \\ &= (x_1 z_1 + x_2 z_2)^2 \\ &= x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= (x_1^2, \sqrt{2}x_1 x_2, x_2^2)(z_1^2, \sqrt{2}z_1 x_2, z_2^2)^T \\ &= \phi(x)^T \phi(z) \end{aligned}$$

after separating x and z we find that

$$\phi(x) = (x_1^2, \sqrt{2}x_1 x_2, x_2^2)^T$$

10 Gaussian Processes

11 Support Vector Machines

11.1 Comparison to Perceptron

Perceptron:

- linear separator
- simple update rule
- prone to overfitting

SVM:

- unique max-margin linear separator
- quadratic optimization
- robust to overfitting