SML lecture notes

Statistical Machine Learning

These are the notes for Statistical Machine Learning. The first lecture had a slide which detected health of a pig using it's picture. (Future prospects: Hiring/Admissions LOL). Anyways, I am using Obsidian and this is an amazing markdown editor! It has a lot of community plugins. Cool, let's get started!

Index

1. SML/Lecture 1 : Introduction to the course and grading.
2. SML/Lecture 2 : Something more here
3. SML/Lecture 3 : Moreeee!
4. Time warp, BDT, MLE, θ_{MAP},PCA, FDA
5. SML/Lecture 13 : Regression
6. SML/Lecture 14 : Bias, Variance, Gaussian kernel
7. SML/Lecture 15: Regularization, beginning of DL
8. SML/Lecture 16: Gradient dissent
9. Time warp, backpropagation, logistic regr.
10. CNN (kernel convolutions) /DL begin etc.
11. SML/Lecture 18: CNN continued, Convolutinal auto-encoders, Decision trees
12. SML/Lecture 19: Live

Lecture sigmoid(infinity)

Below is a big overview (But I already know everything here lol so nothing "new")

Classification

Predicting a discrete random variable Y from another random variable X.

- Consider data $(X_1, Y_1), \ldots, (X_n, Y_n)$ where $X_i = (X_{i1}, X_{i2}, \ldots, X_{id}) \in \mathcal{X} \subset \mathbb{R}^d$
  is a d-dimensinoal vector and Y_i takes values in some finite set $\mathcal{Y}$. A **classification rule** is a function $h : \mathcal{X} \to \mathcal{Y}$. When we observe a new X we can predict Y to be $h(X)$.
- $Y = \{0, 1\}$ binary classification , rest maybe named as multiclass classification

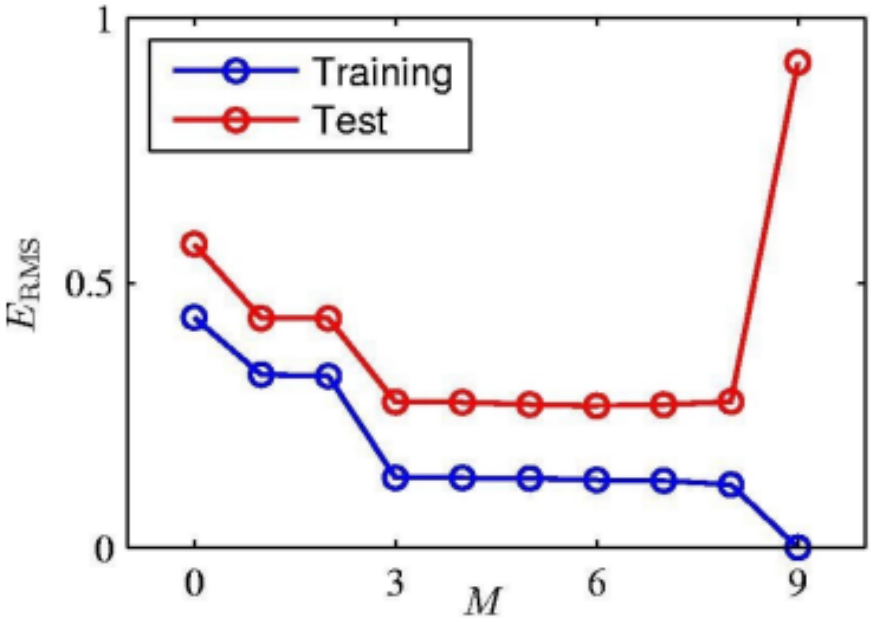
Loss Function

Say $y(x, \vec{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$

- $E(w) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, w) - t_n\}^2$: Sum of squares error function

Over-fitting

When training loss function is low but on testing it becomes high.



Regularization

This is just adding a term in the loss function to penalize when the magnitude of $\vec{w}$ is high. There is Lasso and Ridge regression (L1, L2). Lasso has sum of absolute values instead of sum of magnitude. Infact you may define your very own lol.

- $\widetilde{E}(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2 + \frac{\lambda}{2}||w||^2$

Remaining class on reviing probability (class 10th level lmao)

Reference books

- Hastie, Tibshirani, Friedman Elements of Statistical Learning
- Murphy Machine Learning: a Probabilistic Perspective
- Duda: Pattern Classification

Evaluation

- Assignment (50%) - 5, This is pretty pog, I love the course ig
- Quiz (20%) - 3
- Midsem (15%)
- Endsem (15%)
- All mandatory

Grade cutoffs

- 91-100 A/A+ : Stupid course smh, hate the course (unless jsksksks)
- 81-90 A-
- 71-80 B

Further Reading

- Theoretical : AISTATS, ICML, JMLR, NeurIPS
- Systems+Theory: CVPR, ICCV, ECCV, AAAI, IEEE Transactions

L2 (Regularisation)

Unsupervised learning

Only data, no labels. Example PCA (dim reducition), K-means *clustering*

Looks like nothing was done here apart from revising probability lol.

PSD: Positive semi definite: Hermitian matrix with all eigenvalues positive.

Hermitian matrix when $A = \overline{A^T}$. (complex nos.)

Lecture tHr33

More revision.

Covariance

$$cov[x, y] = \mathbb{E}_{x,y}[\{x - \mathbb{E}[x]\}\{y^T - \mathbb{E}[y^T]\}]$$

so we define it for only one variable X, $cov(X) = \frac{1}{N-1}\sum_{i=1}^{N}[X_i - \mu_X][X_i - \mu_X]^T$ where $\mu_X = \mathbb{E}[x] \in \mathbb{R}^{d\times 1}$

The Gaussian Distribution

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Looks like everyone is a fan of `$\mathcal$` Here, $\mathbb{E}[x] = \mu, var[x] = \sigma^2$

But in this non-binary world there are a lot of things. Presenting multivariate Gaussian (duh)

$$\mathcal{N}(x|\mu, \Sigma^2) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{\overbrace{(x-\mu)^T}^{1\times d}\Sigma^{-1}(x-\mu)}{2}}$$

where obviously $\Sigma = cov(X)$

When $d > N, \Sigma$ is not a full rank matrix (max N). So Σ^{-1} PSD.

- $r^2 = (x-\mu)^T\Sigma{-}1(x-\mu)$ is called a Mahalanobis distance from x to μ. Imagine.
- volume of hyperellipsoid corresponding to a Mahalanobis distance r $V = V_d|\Sigma|^{\frac{1}{2}}r^d$ where V_d is the volume of a d-dimensional unit-hemisphere.
- Higher the determinant for a fixed r and d, higher the scatter. For covariance matrices of independent variables, the determinant is large and thus scatter is more.

Idk what's happening anymore lol.

Now we will do **Bayesian Decision Theory**

Regression

Given $(\mathbf{x}_i, y_i)$ where $\mathbf{x}_i = (x_1, x_2 ...)$ and $y_i \in \mathbb{R}$ a label for a data point. We need to find the function $\hat{f} \approx f : X \mapsto Y$

Linear regression

Say $\hat{f}(x) = w_1x + w_0$, then $\hat{y}_i = w_1x_i + w_0$ the predicted value from the new function, then make $\hat{y}_i \approx y_i$

So, $err_i = \phi(\hat{y}_i, y_i)$ where ϕ be distance or divergence We already know, for ϕ $\phi(a,b) = \phi(b,a), \phi(a,a) = 0$ and triangle ineq. For **divergence** $\phi(a,b)$ may or may not be $\phi(b,a)$

$\hat{Y} = X^T W$ where, $X = \begin{bmatrix} x_1 & x_2 \dots & x_n \\ 1 & 1 \dots & 1 \end{bmatrix}$, $W = \begin{bmatrix} w_1 \\ w_0 \end{bmatrix}$ and $\hat{Y} = \begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ \vdots \\ \hat{y_n} \end{bmatrix}$ and $Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$. We get $\phi(Y, \hat{Y}) = ||Y - \hat{Y}||_2^2$, i.e. l_2 norm. the sum of squares of all values of vector. The l_1 norm is sum of all abs. values in the vector.

Therefore we have $\phi(Y, \hat{Y}) = (Y - \hat{Y})^T (Y - \hat{Y}) = Y^T Y - Y^T \hat{Y} - \hat{Y}^T Y + \hat{Y}^T \hat{Y}$

Now substitute $\hat{Y}$ and minimize w.r.t. W (by getting derivative = 0) Finally we'd get $W = (X^T X)^{-1} XY$

But how far will linear algebra help :trollblob:

Question: What if you take several sub-datasets, compute $\hat{f}_k$ for each of them and compute the average?

$\hat{f}_{avg}(x) = x^2 E(w_2) + xE(w_1) + E(w_0)$ where E is for expectation. $M = 2$ the degree of polynomial.

Bias-Variance Tradeoff

- For higher M: $\hat{f}_{avg}$ can fit into data that require degree M or less.
- Variance $\frac{1}{n-1} \sum (\hat{f}_i(x) - E(\hat{f}))^2$ is large
- $\hat{f}_i$ may be quadratic meanwhile the expectation may be quad, linear or const.
- Bias: Will be small as $E(\hat{f})$ approximately equals true f
- For lower M, we observe underfitting.
- But variance small (less degree of freedom for functions xD) and bias large (linear could never model quadratic).

- Consider degree m = 1, 2, to M
  - Divide data into K folds. K = 5, 10.
    - Hold one fold and use "remaining folds".
    - Learn W from "remaining folds".
    - Apply W to compute error on "remaining
      – call this training error (k)
    - Apply W to compute error on "held out fo
      call this validation error (k)

end

avgTrainErr(m) = mean{train error}

avgValErr(m) = mean{val error}

- end

We use cross-validation to figure out best M.

Gaussian Process

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} K_{11} & \cdots & \\ \vdots & \ddots & \\ & & K_{nn} \end{bmatrix} \right)$$

Similarity function $K(a, b)$

$$K(x_i, x_j) = K_{ij} = \sigma^2 e^{\frac{-\|x_i - x_j\|_2^2}{2l^2}}$$

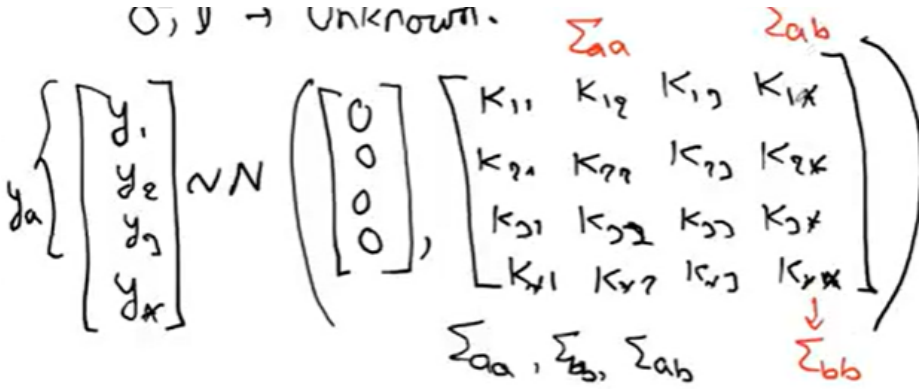
Here l is the width of the gaussian kernel, σ max of kernel, both unknown.

$P(y^*|x, y) \sim \mathcal{N}(\mu^*, \sigma^*)$ where we want the prediction for y^* This is a conditional

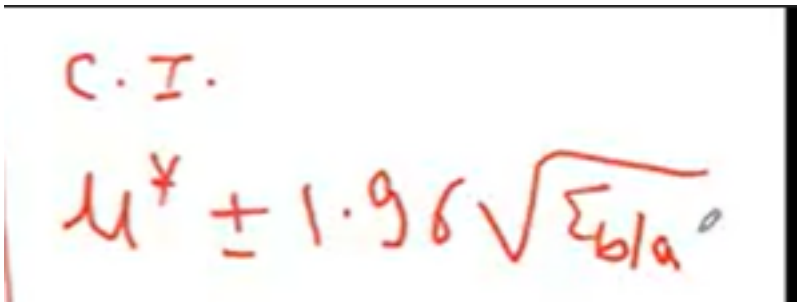
$$\mu_{b|a} = \mu_b + \Sigma_{ba}\Sigma_{aa}^{-1}(y_a - \mu_a)$$

$$\Sigma_{b|a} = \Sigma_{bb} - \Sigma_{ba}\Sigma_{aa}^{-1}\Sigma_{ab}$$

distro on y^* given all other info.



Last y is y^*

So we get a whole ass confidence interval 

Regularization

To reduce overfitting (/reduce model complexity)

L.S.R. (Least squares reg.), we can regularize as

$\min(||Y - X^T W||_2^2 + \alpha W^T W)$ so, now our solution is $W = (X^T X + \alpha I)^{-1} XY$

- Hyperparameter optimize α or use CV?

L_2 regularizer -> Ridge regression, Tikhonov reg, weight decay L_1 -> Lasso regression

Say, $f(w) = ||Y - X^T W||_2^2, g(w) = \alpha W^T W$ so, we minimize $\min_w f(w) + g(w)$ same as $\min_w e^{f(w)} \cdot e^{g(w)}$ exploit monotonicity, take negative $\max(e^{-[f(w)+g(w)]})$

So, the $e^{-g(w)}$ would look like gaussian, whereas incase of L_1 it has a sharp peak but also dies out easily. (May also use as an advantage, feature extraction) -> promotes sparsity

Deep Learning

Perceptron

- Collect input
- Linearly weigh
- add bias
- get the sgn of hole thing (classifier)

Gradient Dissent

$\min_{x,y}(x^2+y^2) = f(x,y)$ We have the gradient $\nabla f = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$ So (n = learning rate), $x_{new} \leftarrow x_{old} - n(\delta f/\delta x)$ and $y_{new} \leftarrow y_{old} - n(\delta f/\delta y)$ i.e. we are going AGAINST the gradient.

Convolutional Auto-Encoders

Encoding: Input, say $I \in \mathbb{R}^{M\times N\times 1} \overset{Conv.(K,(3\times 3))}{\rightarrow} M \times N \times K \overset{activation}{\rightarrow} +\max$ pooling. (maybe 2×2) (to downsample) add more layers with conv blocks and pooling. Note, here K number of kernels/filters exist which are supposed to train.

Decoding: Upsample, maybe ReLu, again upsample till you reach input dims. (obviously)$\rightarrow \hat{I}$ Here, 2×2 upsampling is defined as $[a] \rightarrow \begin{bmatrix} a & a \\ a & a \end{bmatrix}$.

We might use MSE loss. We might also use Spacial Parameter Pooling (SPP) to preserve structural integrity of input.

For the classification part (assuming the previous network is trained), Input -> encode -> flatten -> [learnable] activate + softomax (+multi class cross entropy) **How about we change our objective to something in GAN space (to keep it more identiafiable)**

AE can help when you have limited labelled data.

Model Ensembling (random forest regressors lol)

- Eww Decision trees (but easy to interpret). In theory decision spaces could be arbitary hyper-shapes.
- Since the trees (in practice) divides stuff in high dimensional rectangular regions we can create a loss function like $\sum_{j=1}^{J}\sum_{i\in R}(y_i - \hat{y_R})$
- But that's sad, sad in reverse is das, and das not good.
- Discretize the splits. The split across dimension x_n which gives the lowest error for regionwise class will be the split to go with. For the next split,

you can only choose one among the split regions (random, therefore we use forests or a hive mind lol). Apply iteratively (but thresh it lol, otherwise every sample will have it's own region). Threshing can be like, no. of samples $\geq$ for each class.

- Classification trees (loss Gini Index $G = \sum_{k=1}^{K} \hat{p_k}(1 - \hat{p_k})$ for each node 'm', 'K' is the number of classes and $\hat{p_k}$ is the probability of 'k' in current node.).
  - Binary genie index lol
  - Get dataset D, get n subsets (which may have rep. elements), the remaining (as a result of rep.) shall be used as a validation (also called out of bag OOB error), not much purpose of validation tho, we wont be re-training it, it's just for analysis. We train a forest now. So, given in sample say x^* we shall compute all predictions. Get the majority, ofc. (audience poll xD)
  - bagging uses all features in all trees, in random forest `sqrt` n_features are used (kind of tree independence).

Boosting [XGboost etc/ADAboost in class]

- To boost performance of weak classifiers (which perform slightly better than random.
- We will weigh the misclassfied points (say 1/n where n: sample)
- $D = \{x_i, y_i\}_{i=1}^{n}, y_i \in \{1, 0\}$, and weights $w_i = \frac{1}{n}$. Train, then re-weigh the misclassified. Say 5 misclassied, then weight = 1/5. Now learn again.
- Say we train m such classifiers, then the final $f(x) = sign\left[\sum_{j=1}^{m} \alpha_j h_j(x)\right]$ where $\alpha_j = \frac{1}{2}\log\left(\frac{1-L_j}{L_j}\right)$ and updating weights $w_i \leftarrow w_i \cdot e^{2\alpha_j}$ $\forall$misclassified
- $\forall j \in [1..m]$ learn $h_j \leftarrow \min_H L_j$ where H is the class of the *classifiers*. $L_j = \frac{\sum_{i=1}^{n} w_i \mathbb{I}(y_i \neq h_j(x_i))}{\sum_i w_i}$ the loss where $\mathbb{I}$ (identifier func?) is 1 when classifier is correct, else 0.