Machine Learning - Practice and Theory

Day 8 - SVM, Ensembles, Neural Networks

Govind Gopakumar

IIT Kanpur

Prelude

Announcements

- Doubt clearing session on Thursday (1500 - 1600)
- Programming tutorial on PCA by tomorrow (hopefully)
- Programming tutorial on Kernels (in SVM)

Recap

Principal Components Analysis

- Captures how the data “spreads”
- Means to reduce dimensionality
- Can capture multiple correlations in the data

Kernels

- Increase dimensionality without actually increasing it
- Works if we require only dot products
- Can learn non-linear surfaces with linear methods!

Support Vector Machines

Review of Perceptron

Model overview

- $\langle w, x \rangle > 0.5, < 0.5$: decision rule
- Learnt only a hyperplane
- Could be learnt very fast (stochastic method)

Drawbacks

- What line does it finally learn?
- Can this line be good? Or bad?
- What line should we learn?

SVM - I

Background

- Notion of a “margin”
- When is a line good? When is a line bad?
- Do we increase margin? Or decrease it?

Loss functions

- Perceptron loss : $y_n(\langle w, x_n \rangle) \leq 0$
- Can we modify this?

SVM - II

Model overview

- Learn a “max margin” line
- Correctly classifies both classes
- Classify them with some distance

Time complexity?

- What vectors do I need to specify the model later on?
- “Support Vectors” : how did this come?

SVM - III

Learning the SVM?

- A bit more advanced maths than useful right now
- Can be solved exactly in case of seperable data
- If data isn't seperable? : "Soft - Margin"

Using the SVM

- Wherever we have linear seperability
- What if we don't have seperability?

SVM - IV

Concluding remarks

- Extremely popular : You **WILL** see it used everywhere!
- Highly optimized packages available, even in python!
- Active area of research

But it still learns a line?

- Kernel trick!
- Can also be used for regression

Boosting and ensembles

Ensemble models - I

What?

- What is an “ensemble”?
- How do we construct it?
- Different models on same data vs same model on different data?

How?

- Aggregate or voting on predictions
- Stack : predictions as features!
- Levels of models!

Ensemble models - II

Bagging

- Create multiple copies of data
- Train similar / same models on these copies
- Aggregate predictions

Why would this work?

- Each model captures the variance of data
- Noise is spread out, reduced
- How many replications?

Ensemble models - III

Boosting

- Use only weak algorithms
- Combine them iteratively to get better predictions
- Increase weight of hard examples

Process

- Have T different “weak” models
- Start with uniform weights for all points
- Learn an initial model
- Iteratively increase weights depending on previous mistakes
- Learn better models

Ensemble models - IV

AdaBoost

- Choose a weak model (Perceptron?)
- Let it learn from data
- Check where it made errors : increase these points!
- Choose another perceptron, learn on new data

Can it learn complicated shapes?

- Yes, at times
- Outliers can screw it up a lot at times

Ensemble models - V

Comments

- Bagging vs Boosting? : no real winner
- Bagging allows parallel learning
- Boosting keeps decreasing training error

Why should we use either?

- Reduce overfitting (multiple models)
- Combine predictions

Neural Networks

Review of Perceptron

Model overview

- $\langle w, x \rangle > 0.5, < 0.5$: decision rule
- Learnt only a hyperplane
- Could be learnt very fast (stochastic method)

How did we learn this?

- Stochastic Gradient descent
- $w^{t+1} = w^t - l'(w^t)$

Multi layer Perceptron - I

Structure

- Input layer : Where you feed in data
- Output layer : Where you get output (class, value)
- Hidden layer : Middle “layers”

Zooming in

- All layers : Composed of nodes
- All nodes : Composed of simple perceptron
- Activation : Perceptron output is “activated”

Multi layer Perceptron - II

Computation

- Each node is like a neuron
- Computes a weighted sum of its inputs
- “Activates” it

Use

- One hidden layer is enough to approximate any function!
- Chain together “deep” and “wide” networks

Multi layer Perceptron - III

Feature extraction

- Lower layers “learn” smaller features
- Higher layers “learn” larger features

Activation functions

- Normal perceptron uses step function (0-1)
- Sigmoid function
- ReLU function
- Why are non-linear functions needed?

Multi layer Perceptron - IV

Feedforward

- Input is passed through the network, layer by layer
- $x \to f(W^T x) \to g(V^T f(W^T x)) \to \dots$
- Weighted sum followed by activation

Large networks

- Deep : Multiple layers
- Wide : Multiple nodes within layers

Multi layer Perceptron - V

How do we learn this?

- What are the unknowns?
- What is the loss function?
- Can we do gradient descent?

Backpropagation

- Push errors / updates through the network
- Use “chain rule” from calculus to derive gradients!

Multi layer Perceptron - VI

Backpropagation

- Compute gradients starting from last layer
- Compute next set of gradients step by step

Example network

- Regression problem : $y = f(X)$
- Two layers : $y_i = V^T f(W^T x_i)$

Multi layer Perceptron - VII

Backpropagation of errors

- Loss function : $\min_{W,V} \sum \left(y_i - V^T f(W^T x_i)\right)^2$
- Let us look at it as : $\sum (y_i - \langle V, h_i \rangle)^2$
- Gradient w.r.t V is easy : $\frac{\partial L}{\partial V} = -2 \sum (y_i - \langle V, h_i \rangle) h_i$
- Simple form : $-2 \sum e_n h_n$

Chain rule

- We now need to update W as well - Chain rule
- $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial W}$
- $\frac{\partial L}{\partial f_k} = -\sum e_n v_k$
- $\frac{\partial f}{\partial W} = \sum f'(w_k^t x_n)$

Multi layer Perceptron - VII

Backpropagation

- Compute forward pass : error at output layer
- Compute backward pass : gradients at each layer
- Update parameter at each backward pass

Computational issues

- Update of parameter at layer i depends on $i + 1$
- Parallel updates if we have multiple nodes!

Multi layer Perceptron - VIII

So why the hype?

- Chaining layers increases power
- Extremely fast computation on GPUs
- Extremely powerful structures can be learnt!

Is the hype justified?

- Overall loss is extremely non-convex
- Sometimes blind usage is promoted!
- Theory is not very well developed

Conclusion

Concluding Remarks

Takeaways

- Combining different methods
- Boosting, Bagging
- SVM : Powerful lines!
- MLP : The most powerful machine learning tool known to us!

Announcements

- Doubt clearing session tomorrow : come with doubts!
- Tutorials for SVM + Kernel up
- Will put up tutorial for Kernels in other methods

References

- Lecture 11, CS 771 IIT Kanpur
- Lecture 9, CS 771 IIT Kanpur
- Lecture 21, CS 771 IIT Kanpur