

Supervised Learning (COMP0078)

1. Introduction to supervised learning

Mark Herbster

4 October, 2021

University College London

Department of Computer Science

SLIntro-21v4

References

Useful References

- *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, T. Hastie, R. Tibshirani, & J. Friedman, Springer (2009)
- *Understanding Machine Learning from Theory to Algorithms*, S. Shalev-Shwartz and S. Ben-David, Cambridge University Press (2014)
- *Kernel Methods for Pattern Analysis*, J. Shawe-Taylor, and N. Cristianini, Cambridge University Press (2004)
- *Pattern Recognition and Machine Learning*, C. Bishop, Springer (2006)

Course information

1. When

- Monday 4-5 pm [Lecture]
- Tuesday 5-6 pm [Lecture]
- Wednesday 4-5 pm [Office Hours]
- Thursday 1-2 pm [TA-Session 1, OPTIONAL]
- Friday 12-1 pm [TA-Session 2, OPTIONAL]
- Friday 1-2 pm [TA-Session 3, OPTIONAL]
- Friday 5-6 pm [Lecture]
- Note : NO TA sessions (Week 1)

2. Questions :

2.1 Moodle forum

2.2 sl-support@cs.ucl.ac.uk

Assessment

1. Coursework (50%) and Exam (50%)
2. 2 courseworks assignments
(deliver them on-time, penalty otherwise)
3. To pass the course, you must obtain an average of at least 50% when the homework and exam components are weighted together.

Prerequisites

- Calculus (real-valued functions, limits, derivatives, Taylor series, integrals,...)
- Elements of probability theory (random variables, expectation, variance, conditional probabilities, Bayes rule,...)
- Fundamentals of linear algebra (vectors, angles, matrices, eigenvectors/eigenvalues,...),
- A bit of optimization theory (convex functions, Lagrange multipliers)

Provisional course outline

- (Lecture 1) Key concepts (probabilistic formulation of learning from examples, loss function, learning algorithm, overfitting and underfitting, model selection, cross validation); two basic learning algorithms linear regression and k -NN;
- (Lecture 2) Regularisation, Kernels
- (Lecture 3) Support Vector Machines
- (Lecture 4) Decision Trees and Ensemble Learning
- (Lecture 5) Online Learning I

Provisional course outline

- (Lecture 6) Graphs in Machine Learning
- (Lecture 7) Learning Theory
- (Lecture 8) Online Learning II
- (Lecture 9) Matrix Completion (time permitting)

This week's plan

- The supervised learning problem
- Two learning algorithms: least squares and k -NN
- Probabilistic model, error function, optimal solutions
- Bias-Variance tradeoff, NFL theorems
- Asymptotic Optimality of k -NN
- Hypothesis space, overfitting and underfitting
- Choice of the learning algorithm (Model selection)

Supervised Learning Problem

Given a set of **input/output** pairs (**training set**) we wish to compute the functional relationship between the input and the output

$$\mathbf{x} \longrightarrow \boxed{f} \longrightarrow y$$

- **Example 1:** (people detection) given an image we wish to say if it depicts a person or not. The output is one of 2 possible categories
- **Example 2:** (pose estimation) we wish to predict the pose of a face image The output is a continuous number (here a real number describing the face rotation angle)

In both problems the input is a high dimensional vector \mathbf{x} representing pixel intensity/color

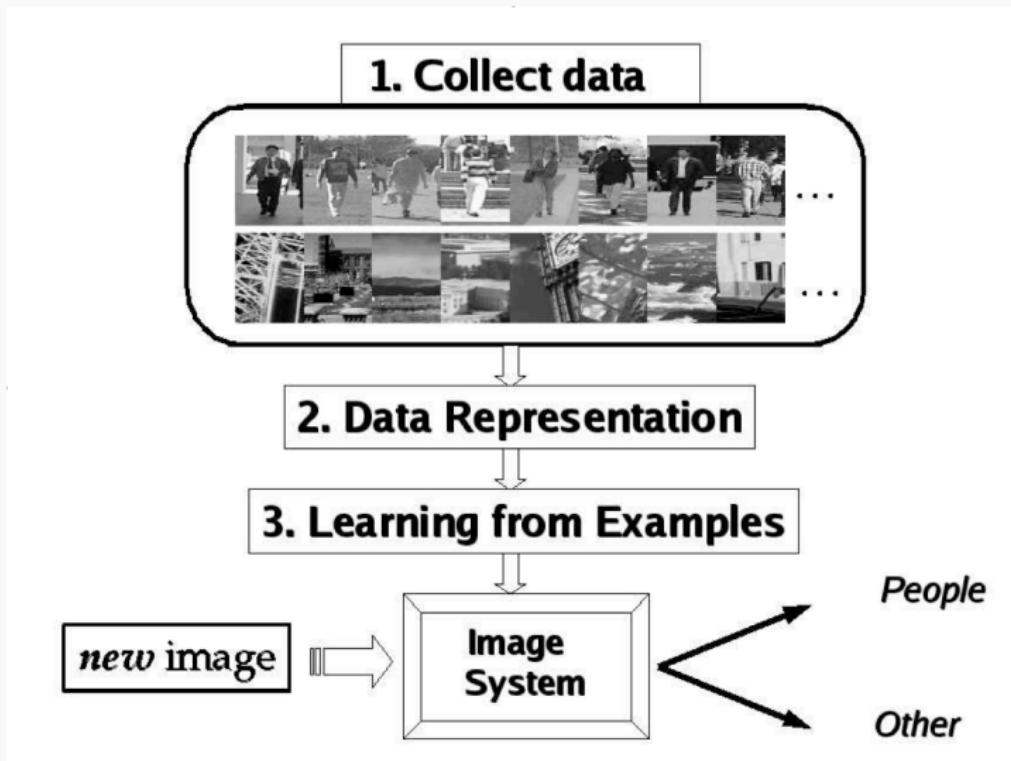
Why and when learning?

- The aim of learning is to develop software systems able to perform particular tasks such as people detection.
- Standard software engineering approach would be to specify the problem, develop an algorithm to compute the solution, and then implement efficiently.
- Problem with this approach is developing the algorithm:
 - No known criterion for distinguishing the images;
 - In many cases humans have no difficulty;
 - Typically problem is to specify the problem in logical terms.

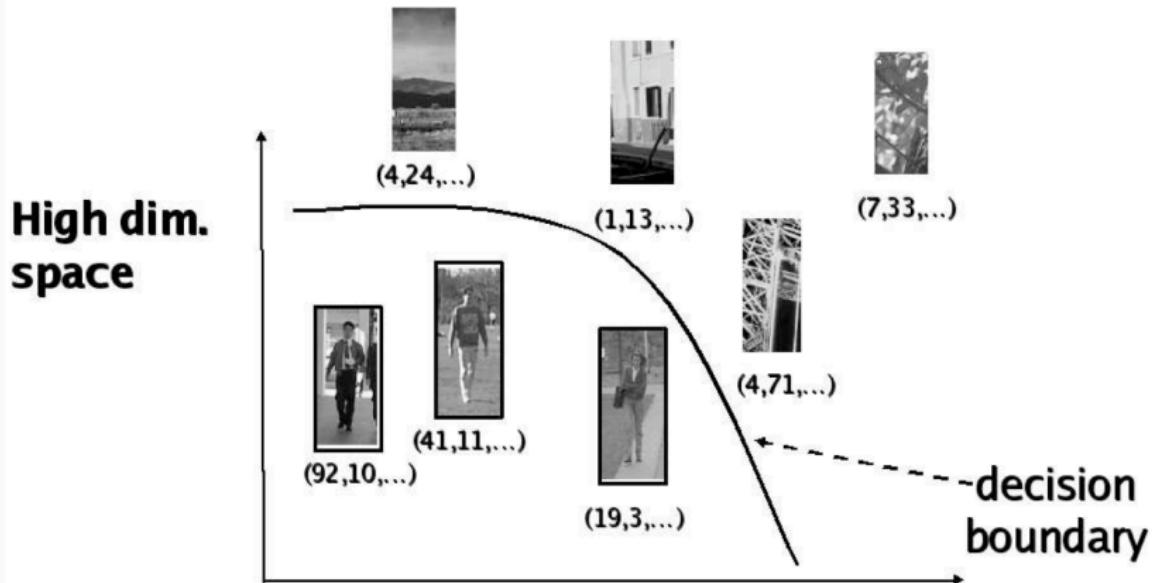
Learning approach

- Learning attempts to infer the algorithm for a set of (labelled) examples in much the same way that children learn by being shown a set of examples (eg sports/non sports car).
- Attempts to isolate underlying structure from a set of examples. Approach should be
 - stable: finds something that is not chance part of set of examples
 - efficient: infers solution in time polynomial in the size of the data
 - robust: should not be too sensitive to mislabelled/noisy examples

People Detection Example



People detection example (cont.)



Data are sparse! Risk for overfitting!

Supervised Learning Model

- Goal: Given training data (pattern,target) pairs

$$S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$$

infer a function f_S such that

$$f_S(\mathbf{x}_i) \approx y_i$$

for the **future** data

$$S' = \{(\mathbf{x}_{m+1}, y_{m+1}), (\mathbf{x}_{m+2}, y_{m+2}), \dots\}.$$

- Classification : $y \in \{-1, +1\}$; Regression : $y \in \mathbb{R}$
- \mathcal{X} : input space (eg, $\mathcal{X} \subseteq \mathbb{R}^d$), with elements $\mathbf{x}, \mathbf{x}', \mathbf{x}_i, \dots$
- \mathcal{Y} : output space, with elements y, y', y_i, \dots

Supervised learning problem: compute a function which “best describes” I/O relationship

Learning algorithm

- Training set: $S = \{(\mathbf{x}_i, y_i)_{i=1}^m\} \subseteq \mathcal{X} \times \mathcal{Y}$
- A **learning algorithm** is a mapping $S \mapsto f_S$
- A new input \mathbf{x} is predicted as $f_S(\mathbf{x})$

Example Algorithms

- Linear Regression
- Neural Networks
- Decision Trees
- Support Vector Machines

- In the course we mainly deal with deterministic algorithms but we'll also comment on some randomized ones
- Today: we describe two simple learning algorithms:
linear regression and k -nearest neighbours

Some Questions

- How is the data **collected**? (need assumptions!)
- How do we **represent** the inputs? (may require preprocessing step)
- How **accurate** is f_s on new data (study of **generalization error**)?
- Many algorithms may exist for a task. How do we choose?
- How “**complex**” is a learning task? (computational complexity, sample complexity)

Data Ethics [Not Examined]

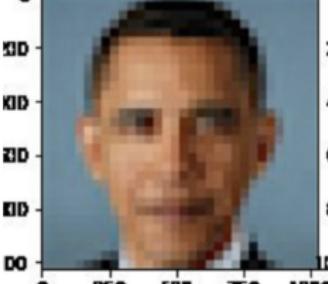
The New York Times

There Is a Racial Divide in Speech-Recognition Systems, Researchers Say

Technology from Amazon, Apple, Google, IBM and Microsoft misidentified 35 percent of words from people who were black. White people fared much better.



Amazon's Echo device is one of many similar gadgets on the market. Researchers say there is a racial divide in the usefulness of speech recognition systems. Grant Hindsley for The New York Times



Original



Result

For an introductory discussion see this talk by Emily Denton & Timnit Gebru

Slides: <https://drive.google.com/file/d/1IvUgCTUciIJQ-dIqQAYN011X3guzqnYN/view>

Talk: https://youtu.be/v_XBJd1Fxqc

Some difficulties/aspects of the learning process

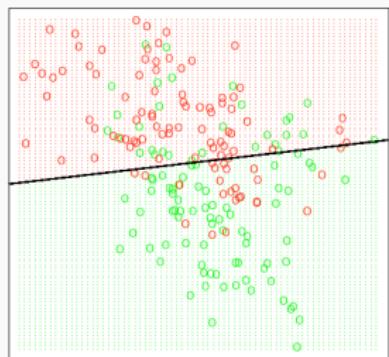
- New inputs **differ** from the ones in the training set (look up tables do not work!)
- Inputs are measured with **noise**
- Output is **not deterministically** obtained by the input
- Input is often **high dimensional** but some components/variables may be irrelevant
- How can we incorporate **prior knowledge?**

Binary classification: an example

We describe two basic learning algorithms/models for classification which can be easily adapted to regression as well.
We choose: $\mathcal{X} = \mathbb{R}^2$, $\mathbf{x} = (x_1, x_2)$ and $\mathcal{Y} = \{\text{green}, \text{red}\}$

Our first learning algorithm computes a **linear function**, $\mathbf{w}^\top \mathbf{x} + b$ and classifies an input \mathbf{x} as

$$f(\mathbf{x}) = \begin{cases} \text{red} & \mathbf{w}^\top \mathbf{x} + b > 0 \\ \text{green} & \mathbf{w}^\top \mathbf{x} + b \leq 0 \end{cases}$$



Linear Regression (Least Squares)

- Emerged in response to problems in Astronomy and Navigation
- Motivated by the need to combine multiple noisy measurements
- Method first described by Gauss in 1794



A Simple Problem – 1

Given the data set

$$S = \{((1, 1), 3), ((2, 3), 7)\}$$

Then with the new input $\mathbf{x}_3 = (4, 2)$

how should we predict y_3 ?

Why? What Assumptions?

A Simple Problem – 2

Model as a system of equations

$$w_1 + w_2 = 3$$

$$2w_1 + 3w_2 = 7$$

or more directly as

$$\mathbf{X}\mathbf{w} = \mathbf{y}$$

with

$$\mathbf{X} = \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}; \quad \mathbf{y} = \begin{pmatrix} 3 \\ 7 \end{pmatrix}$$

A Simple Problem – 3

Solving in matlab

$$\mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$$

```
>> y = [3 ; 7]
y = 3
      7
>> X = [1,1 ; 2, 3]
X = 1      1
      2      3
>> XI = X^(-1)
XI = 3      -1
      -2      1
>> w= XI * y
w = 2
      1
>> w= X \ y %% More efficient than calculating inverse use in practice see help mldivide
w = 2
      1
```

We now have the linear predictor

$$\hat{y} = \mathbf{w} \cdot \mathbf{x}$$

Thus predict $\hat{y}_3 = \mathbf{w} \cdot \mathbf{x}_3 = w_1 x_{3,1} + w_2 x_{3,2} = 4 \times 2 + 1 \times 2 = 10.$ 23

A Simple Problem – 4

What if?

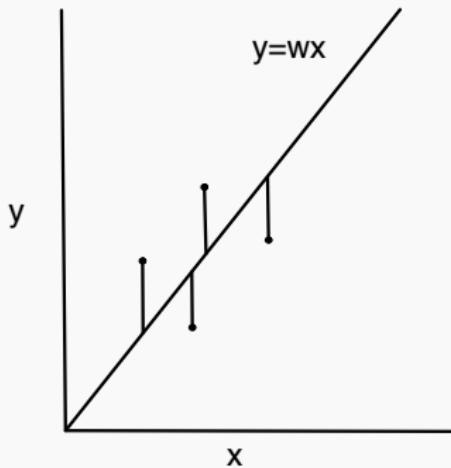
- Overdetermined:

$$S = \{((1, 1), 3), ((2, 3), 7), ((2, 1), 3)\}$$

- Underdetermined:

$$S = \{((1, 1, 2), 3), ((2, 4, 3), 7)\}$$

Minimize square error – 1



Find a linear predictor $\hat{y} = \mathbf{w} \cdot \mathbf{x}$ to minimize the square error over the data $\mathcal{S} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$ thus

$$\text{Minimize: } \sum_{i=1}^m (y_i - \hat{y}_i)^2 = \sum_{i=1}^m (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2$$

Minimize square error – 2

Thus given,

$$\mathcal{S} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$$

with $\mathbf{x} \in \mathbb{R}^n$ we may represent the pattern and target vectors with the matrices

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & \dots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \dots & x_{m,n} \end{pmatrix} \text{ and } \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

MSE in Matrix Notation

Thus in matrix notation the *empirical* mean (square) error of the linear predictor $\hat{y} = \mathbf{w} \cdot \mathbf{x}$ on the data sequence \mathcal{S} is

$$\begin{aligned}\mathcal{E}_{\text{emp}}(\mathcal{S}, \mathbf{w}) &= \frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2 \\ &= \frac{1}{m} \sum_{i=1}^m (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2 \\ &= \frac{1}{m} \sum_{i=1}^m (y_i - \sum_{j=1}^n w_j x_{i,j})^2 \\ &= \frac{1}{m} (\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y})\end{aligned}$$

MSE minimization. General Case

To compute the minimum we solve for

$$\nabla_{\mathbf{w}} \mathcal{E}_{\text{emp}}(\mathcal{S}, \mathbf{w}) = \mathbf{0}.$$

recalling that

$$\nabla_{\mathbf{w}} = \begin{pmatrix} \frac{\partial}{\partial w_1} \\ \vdots \\ \frac{\partial}{\partial w_n} \end{pmatrix}$$

Thus we need to solve,

$$\begin{aligned} \nabla_{\mathbf{w}} \left[(\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) \right] &= \mathbf{0} \\ \left(\sum_{i=1}^m \frac{\partial}{\partial w_1} \left(\sum_{j=1}^n X_{ij} w_j - y_i \right)^2, \dots, \sum_{i=1}^m \frac{\partial}{\partial w_n} \left(\sum_{j=1}^n X_{ij} w_j - y_i \right)^2 \right)^T &= \mathbf{0} \end{aligned}$$

Normal equations

Consider the 2-d case ($n = 2$)

$$\mathcal{E}_{\text{emp}}(\mathcal{S}, \mathbf{w}) = \frac{1}{m} \sum_{i=1}^m (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$$

Note that:

$$\frac{\partial \mathcal{E}_{\text{emp}}(\mathcal{S}, \mathbf{w})}{\partial w_k} = \frac{2}{m} \sum_{i=1}^m (\mathbf{w}^\top \mathbf{x}_i - y_i) \frac{\partial (\mathbf{w}^\top \mathbf{x}_i)}{\partial w_k} = \frac{2}{m} \sum_{i=1}^m (\mathbf{w}^\top \mathbf{x}_i - y_i) x_{ik}$$

Hence, to find $\mathbf{w} = (w_1, w_2)^\top$ we need to solve the *linear system* of equations

$$\sum_{i=1}^m (x_{ik} x_{i1} w_1 + x_{ik} x_{i2} w_2) = \sum_{i=1}^m x_{ik} y_i, \quad k = 1, 2$$

Normal equations (cont.)

In vector notations:

$$\sum_{i=1}^m \mathbf{x}_i \mathbf{x}_i^\top \mathbf{w} = \sum_{i=1}^m \mathbf{x}_i y_i$$

In matrix notation:

$$\mathbf{X}^\top \mathbf{X} \mathbf{w} = \mathbf{X}^\top \mathbf{y}$$

where

$$\mathbf{X}^\top = \begin{bmatrix} x_{11} & \cdots & x_{m1} \\ \vdots & \ddots & \vdots \\ x_{1n} & \cdots & x_{mn} \end{bmatrix} \equiv [\mathbf{x}_1, \dots, \mathbf{x}_m], \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

Least square solution

$$X^T X \mathbf{w} = X^T \mathbf{y}$$

For the time being we will assume that the matrix $X^T X$ is invertible, so we conclude that

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

Otherwise, the solution may not be unique...

Comment:

In matlab use

$$\mathbf{w} = X \backslash \mathbf{y}$$

Going back to “b” (adding a bias term)

Substituting \mathbf{x}^\top by $(\mathbf{x}^\top, 1)$ and \mathbf{w}^\top by (\mathbf{w}^\top, b) , the above system of equations can be expressed in matrix form as (exercise):

$$(\mathbf{X}^\top \mathbf{X})\mathbf{w} + \mathbf{X}^\top \mathbf{1}b = \mathbf{X}^\top \mathbf{y}$$

$$\mathbf{1}^\top \mathbf{X}\mathbf{w} + mb = \mathbf{1}^\top \mathbf{y}$$

that is

$$\begin{bmatrix} \mathbf{X}^\top \mathbf{X} & \mathbf{X}^\top \mathbf{1} \\ \mathbf{1}^\top \mathbf{X} & m \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{X}^\top \mathbf{y} \\ \mathbf{1}^\top \mathbf{y} \end{bmatrix}$$

where $\mathbf{1} = (1, 1, \dots, 1)^\top$, an $m \times 1$ vector of “ones”

A different approach: k -nearest neighbours

Let $N(\mathbf{x}; k)$ be the set of k nearest training inputs to \mathbf{x} and

$$I_{\mathbf{x}} = \{i : \mathbf{x}_i \in N(\mathbf{x}; k)\}$$

the corresponding index set

$$f(\mathbf{x}) = \begin{cases} \text{red} & \text{if } |\{y_i = \text{red} : i \in I_{\mathbf{x}}\}| > |\{y_i = \text{green} : i \in I_{\mathbf{x}}\}| \\ \text{green} & \text{if } |\{y_i = \text{green} : i \in I_{\mathbf{x}}\}| > |\{y_i = \text{red} : i \in I_{\mathbf{x}}\}| \end{cases}$$

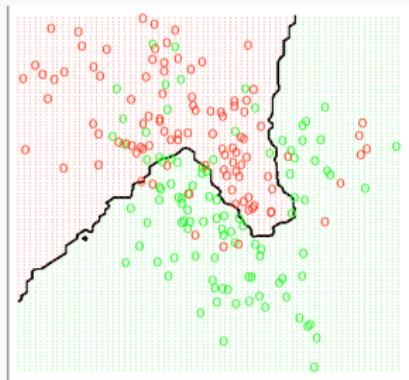
- Closeness is measured using a metric (eg, Euclidean dist.)
- Local rule (compute local majority vote)
- Decision boundary is non-linear

Note: for regression we set $f(\mathbf{x}) = \frac{1}{k} \sum_{i \in I_{\mathbf{x}}} y_i$ (a “local mean”)

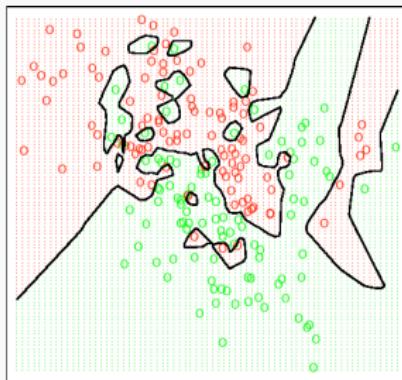
k -NN: the effect of k

- The smaller k the more irregular the decision boundary

$k = 15$



$k = 1$



- How to choose k ? later...

Perspectives on supervised learning

Overview

We consider three influential ideas that underpin SL.

1. What is an optimal predictor? The Bayes estimator.
2. The bias and variance of a learning algorithm.
3. The non-existence of an optimal learning algorithm (without assumptions). NFL theorems.

Optimal Supervised Learning

Model: We assume that the data is obtained by sampling **i.i.d.** from a **fixed but unknown** probability distribution $P(\mathbf{x}, y)$

Expected error (wrt to the *squared loss*):

$$\mathcal{E}(f) := \mathbf{E} \left[(y - f(\mathbf{x}))^2 \right] = \int (y - f(\mathbf{x}))^2 dP(\mathbf{x}, y)$$

Our goal is to minimize \mathcal{E}

Optimal solution: $f^* := \operatorname{argmin}_f \mathcal{E}(f)$ (called *Bayes estimator*)

Problem A: in order to compute f^* we need to know P !

Note: for binary classification with $\mathcal{Y} = \{0, 1\}$ and $f : \mathcal{X} \rightarrow \mathcal{Y}$, $\mathcal{E}(f)$ counts the average number of mistakes of f (aka expected misclassification error)

Bayes estimator for square loss

Let us compute the optimal solution f^* for regression $\mathcal{Y} = \mathbb{R}$.

Using the decomposition $P(y, \mathbf{x}) = P(y|\mathbf{x})P(\mathbf{x})$ we have

$$\mathcal{E}(f) = \int_{\mathcal{X}} \left\{ \int_{\mathcal{Y}} (y - f(\mathbf{x}))^2 dP(y|\mathbf{x}) \right\} dP(\mathbf{x})$$

Observe that f^* is

$$f^*(\mathbf{x}) = \int_{\mathcal{Y}} y dP(y|\mathbf{x})$$

Deriving the Bayes estimator for square loss

Deriving f^* in the discrete case,

$$\mathcal{E}(f) := \sum_{x \in X} \sum_{y \in Y} (y - f(x))^2 p(x, y)$$

and $\mathcal{E}(f)$ is the expected error of an arbitrary predictor f wrt the square loss and $p(x, y)$ is a probability mass function.

Separating

$$\mathcal{E}(f) = \sum_{x \in X} [\sum_{y \in Y} (y - f(x))^2 p(y|x)] p(x)$$

now let's find the value of $f^*(\cdot)$ at a specific point. Define

$$\mathcal{E}(f(x)) := \sum_{y \in Y} (y - f(x))^2 p(y|x)$$

which denotes the expected error of f at x .

Derivation – continued (1)

Observe that,

$$\mathcal{E}(f) = \sum_{x \in X} \mathcal{E}(f(x))p(x)$$

i.e., the expected error of f is just the sum of expected errors from each ' x ' weighted by the probability of each ' x '.

Let's solve for the value of f^* at x' by taking the derivative of $\mathcal{E}(f)$ wrt $f(x')$.

$$\begin{aligned}\frac{\partial \mathcal{E}(f)}{\partial f(x')} &= \sum_{x \in X} \frac{\partial [\mathcal{E}(f(x))p(x)]}{\partial f(x')} \\ &= \frac{\partial [\mathcal{E}(f(x'))p(x')]}{\partial f(x')} + \sum_{x \neq x'} \frac{\partial [\mathcal{E}(f(x))p(x)]}{\partial f(x')} \\ &= \frac{\partial [\sum_{y \in Y} (y - f(x'))^2 p(y|x') p(x')]}{\partial f(x')}\end{aligned}$$

Derivation – continued (2)

Set $z := f(x')$ and rewriting gives

$$\begin{aligned}\frac{\partial \mathcal{E}(f)}{\partial z} &= \frac{\partial [\sum_{y \in Y} (y - z)^2 p(y|x') p(x')]}{\partial z} \\ &= -2 \sum_{y \in Y} (y - z) p(y|x') p(x')\end{aligned}$$

Setting equal to zero and solving,

$$0 = \left(\sum_{y \in Y} y p(y|x') - \sum_{y \in Y} z p(y|x') \right) \cancel{p(x')} = \sum_{y \in Y} y p(y|x') - z$$

which implies

$$z = \sum_{y \in Y} y p(y|x')$$

and hence since x' was generic

$$f^*(x) = \sum_{y \in Y} y p(y|x) = E[y|x].$$

Bias and Variance of a learning algorithm

- We now additionally assume there exists some underlying function F such that

$$y = F(x) + \epsilon$$

where ϵ is white noise, i.e., $E[\epsilon] = 0$ and finite variance.

- Thus the optimal prediction is $f^*(x) := E[y|x] = F(x)$ with square loss.
- We would like to understand the expected error by an arbitrary learner $A_S(x)$ (we now drop the S for convenience).
- Our goal will be to understand the expected error at x'

$$\mathcal{E}(A(x')) = E[(y' - A(x'))^2]$$

where y' is a sample from the marginal $P(y|x')$.

Useful Lemma and Corollary

Lemma

$$E[(Z - E[Z])^2] = E[Z^2] - E[Z]^2$$

Proof

$$E[(Z - E[Z])^2] = E[Z^2 - 2ZE[Z] + E[Z]^2]$$

$$\begin{aligned} E[(Z - E[Z])^2] &= E[Z^2] - 2E[Z]^2 + E[Z]^2 \\ &= E[Z^2] - E[Z]^2 \end{aligned}$$

Corollary

$$E[Z^2] = E[(Z - E[Z])^2] + E[Z]^2$$

Decomposing $\mathcal{E}(A(x'))$

$$\begin{aligned} E[(y' - A(x'))^2] &= E[(y')^2 - 2y'A(x') + A(x')^2] \\ &= E[(y' - f^*(x'))^2] + f^*(x')^2 + \\ &\quad - 2f^*(x')E[A(x')] + \\ &\quad E[(A(x') - E[A(x')])^2] + E[A(x')]^2 \\ &= E[(y' - f^*(x'))^2] + \text{[Bayes error]} \\ &\quad (f^*(x') - E[A(x')])^2 + \text{[bias}^2\text{]} \\ &\quad E[(A(x') - E[A(x')])^2] \text{ [variance]} \end{aligned}$$

- Bayes error : $E[(y' - f^*(x'))^2]$ is the irreducible noise.
- Bias : $f^*(x') - E[A(x')]$
describes the discrepancy between the algorithm and “truth” .
- Variance : $E[(A(x') - E[A(x')])^2]$
captures the variance of the algorithm between training sets

Subtlety: There are two *independent* random quantities. $A(\cdot)$ depends on the random draw of the training sequence and y' depends on the draw on from the marginal $p(y|x')$.

Bias and Variance Dilemma

- The bias and variance tend to trade off against one another
- Many parameters better flexibility to fit the data thus low bias but high variance
- Few parameters give high bias but the fit between different data sets will not change much thus low variance
- **Caveat:** this exact decomposition only holds for the square loss.

Optimal Algorithms and No Free Lunch Theorems

- Is there an optimal machine learning algorithm?
- I.e., an algorithm that does almost as well as the bayes estimator.
- There are a variety of such (NFL) results showing that no such algorithm exists.
- We give an example NFL Theorem in classification setting

Bayes estimator for classification

More generally the *Bayes classifier* (estimator) is the minimiser of the expected loss

- for C-class classification, f^* (called the Bayes classifier) is

$$f^*(\mathbf{x}) = \operatorname{argmax}_{c \in \{1, \dots, C\}} P(Y = c | \mathbf{x})$$

where the loss is 0 if we predict correctly and 1 otherwise.

- The *Bayes error rate* (optimal) is then

$$\int (1 - P(Y = f^*(\mathbf{x}) | \mathbf{x})) dP(\mathbf{x})$$

Challenge: What is the Bayes estimator of

$$\mathcal{E}(f) := \sum_{x \in X} \sum_{y \in Y} |y - f(x)| p(x, y)$$

An NFL theorem for classification

Theorem (Shalev-Shwartz and Ben-David 2014, Section 5.1)

Let A_S be any learning algorithm for binary classification $\mathcal{Y} = \{0, 1\}$ over the domain \mathcal{X} . Let $m < |\mathcal{X}|/2$ then there exists some distribution P over $\mathcal{X} \times \{0, 1\}$ such that,

1. There exists a function $f : \mathcal{X} \rightarrow \{0, 1\}$ the Bayes error is 0.
2. With probability of at least $1/7$ with respect to a random draw S of m examples from P and a draw of a $m + 1$ st example (x', y') we have that

$$\text{Prob}[A_S(x') \neq y'] \geq 1/8$$

Proof Intuitions (Not examined)

Proof Sketch. Theorem (Shalev-Shwartz and Ben-David 2014, Section 5.1)

1. Intuition: if an $f : \mathcal{X} \rightarrow \{0, 1\}$ is chosen uniformly at random. There is no “traction” for any learning algorithm except “memorization”.
2. Hence suppose $m = |\mathcal{X}|/2$.
3. The intuition suggests we will be able to predict accurately on up to 50% of the points and on the remaining points it will only be 50% accurate.
4. Thus

$$E[\text{Prob}[A_{\mathcal{S}}(x') \neq y']] \geq 1/4$$

5. Let $\theta := \text{Prob}[A_{\mathcal{S}}(x') \neq y']$
6. θ is random variable wrt to the draw of the training sequence \mathcal{S}
7. Using Markov's inequality for a random variable $\theta \in [0, 1]$ we have that $\text{Prob}[\theta \geq a] \geq \frac{E[\theta] - a}{1 - a}$.
8. Thus since by the “intuition” $E[\theta] \geq 1/4$ and then if we set $a := 1/8$ we have ...
9. With probability of at least $1/7$ with respect to a random draw \mathcal{S} of m examples from P and a draw of a $m + 1$ st example (x', y') we have that

$$\text{Prob}[A_{\mathcal{S}}(x') \neq y'] \geq 1/8$$

Asymptotic Optimality of k -NN

Revisiting k -NN

k -NN attempts approximate $P(Y = c|\mathbf{x})$ as $\frac{|\{i:y_i=c,i\in I_x\}|}{k}$

- Expectation is replaced by averaging over sample data
- Conditioning at \mathbf{x} is relaxed to conditioning on some region close to \mathbf{x}

As the number of samples goes to infinity ($m \rightarrow \infty$) 1-NN and k -NN become “good” estimators.

1-NN is near asymptotically optimal

Theorem

As the number samples goes to infinity the error rate is no more than twice the Bayes error rate.

Proof Sketch

Abbreviate notation $P(c|\mathbf{x}) := P(Y = c|\mathbf{x})$.

The expected (Bayes) error of the Bayes classifier (at x) is

$$1 - \max_{c \in \{1, \dots, C\}} P(c|\mathbf{x})$$

and the expected rate of 1-NN (at x) is

$$\sum_{c=1}^C P(c|\mathbf{x}_{nn})[1 - P(c|\mathbf{x})].$$

Proof Sketch – continued

Proof Sketch

Observe that as the number samples goes to infinity, $m \rightarrow \infty$,

$$P(c|\mathbf{x}) \approx P(c|\mathbf{x}_{nn})$$

thus the expected rate of 1-NN (at x) is

$$\sum_{c=1}^C P(c|\mathbf{x})[1 - P(c|\mathbf{x})].$$

We need to show

$$\sum_{c=1}^C P(c|\mathbf{x})[1 - P(c|\mathbf{x})] \leq 2[1 - \max_{c \in \{1, \dots, C\}} P(c|\mathbf{x})]$$

Proof Sketch – continued

Proof Sketch – continued

Let $c^* = \operatorname{argmax}_{c \in \{1, \dots, C\}} P(c|\mathbf{x})$ and $p^* = P(c^*|\mathbf{x})$. Observe that

$$\begin{aligned} \sum_{c=1}^C P(c|\mathbf{x})[1 - P(c|\mathbf{x})] &= \sum_{c \neq c^*}^C P(c|\mathbf{x})[1 - P(c|\mathbf{x})] + p^*(1 - p^*) \\ &\leq (C - 1) \frac{1 - p^*}{C - 1} [1 - \frac{1 - p^*}{C - 1}] + p^*(1 - p^*) \\ &= (1 - p^*)[1 - \frac{1 - p^*}{C - 1} + p^*] \end{aligned}$$

Where the second line follows since the sum is maximised when all “ $P(c|\mathbf{x})$ ” have the same value. And since $p^* < 1$ we are done. \square

k -NN is asymptotically optimal

One can show that $\mathcal{E}(k(m) - \text{NN}) \rightarrow \mathcal{E}(f^*)$ as $m \rightarrow \infty$ provided that:

1. $k(m) \rightarrow \infty$
2. $\frac{k(m)}{m} \rightarrow 0$

Weakness: the rate of convergence depends exponentially on the input dimension. An example of the **curse of dimensionality**.

Reference for 1-NN near optimality

Cover & Hart : *Nearest Neighbor Pattern Classification*, 1967

Curse of dimensionality

1. “Curse of dimensionality” is an observation that many methods suffer as the dimensionality of data increases.
 2. The intuition is that since volume increases *exponentially* with the dimension then the data required to “cover” the space to perform estimates also increase exponentially.
 3. E.g., consider the unit d -dimensional ball centered at the origin versus the $1/2$ -unit ball at the origin the ratios of their volumes is $\frac{1}{2}^d$.
- Potential “solutions” include limiting the number of hypotheses (functions) – i.e., introducing a *hypothesis space*. Alternately associating a “complexity” with each hypothesis and then “prefer” simpler hypothesis.

Linear regression vs. k -NN (informal)

- Parametric vs. non-parametric
- Global vs. local
- Linear vs. non-linear
- Bias / variance considerations:
 - LR relies heavily on linear assumption (may have large bias) k -NN does not
 - LR is stable (solution does not change much if data are perturbed) 1-NN isn't!
- k -NN sensitive to input dimension d : if d is high, the inputs tends to be far away from each other!

Hypothesis Space

Solving the “Problem A”

$P(\mathbf{x}, y)$ is unknown \Rightarrow cannot compute $f^* = \operatorname{argmin}_f \mathcal{E}(f)$

We are only given a sample (training set) from P

A natural approach: we approximate the expected error $\mathcal{E}(f)$ by the empirical error

$$\mathcal{E}_{\text{emp}}(\mathcal{S}, f) = \frac{1}{m} \sum_{i=1}^m (y_i - f(\mathbf{x}_i))^2$$

Problem B: If we minimize \mathcal{E}_{emp} over all possible functions, we can always find a function with zero empirical error! (if the Bayes error is zero).

Why is this a problem?

Solving the “Problem B”

A Proposed solution: we introduce a **restricted** space of functions \mathcal{H} called the **hypothesis space**

We minimize $\mathcal{E}_{\text{emp}}(\mathcal{S}, f)$ within \mathcal{H} . That is, our learning algorithm is:

$$f_S = \operatorname{argmin}_{f \in \mathcal{H}} \mathcal{E}_{\text{emp}}(\mathcal{S}, f)$$

This approach is usually called **empirical error (risk) minimization**

For example (Least Squares) :

$$\mathcal{H} = \{f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} : \mathbf{w} \in \mathbb{R}^n\}$$

Problem C: How do we choose a space \mathcal{H} (discuss later)?

Choosing a Hypothesis Space (returning to prob. “C”)

Given the training data $y_i = f^*(\mathbf{x}_i) + \epsilon_i$, the goal is to compute an “approximation” of f^* .

We look for an approximant of f^* within a prescribed hypothesis space \mathcal{H}

- Unless prior knowledge is available on f^* (eg, f^* is linear) we cannot expect $f^* \in \mathcal{H}$
- Choosing \mathcal{H} “very large” leads to **overfitting!** (we’ll see an example of this in a moment)

Summary

- Data S sampled i.i.d from P (fixed but unknown)
- f^* is what we want, f_S is what we get
- Different approaches to attempt to estimate/approximate f^* :
 - Minimize \mathcal{E}_{emp} in some restricted space of functions (eg, linear)
 - Compute local approximation of f^* (k -NN)
 - Estimate P and then use Bayes rule...

Model Selection

Polynomial fitting

As an example of hypothesis spaces of increasing “complexity” consider regression in one dimension

$$H_0 = \{f(x) = b : b \in \mathbb{R}\}$$

$$H_1 = \{f(x) = ax + b : a, b \in \mathbb{R}\}$$

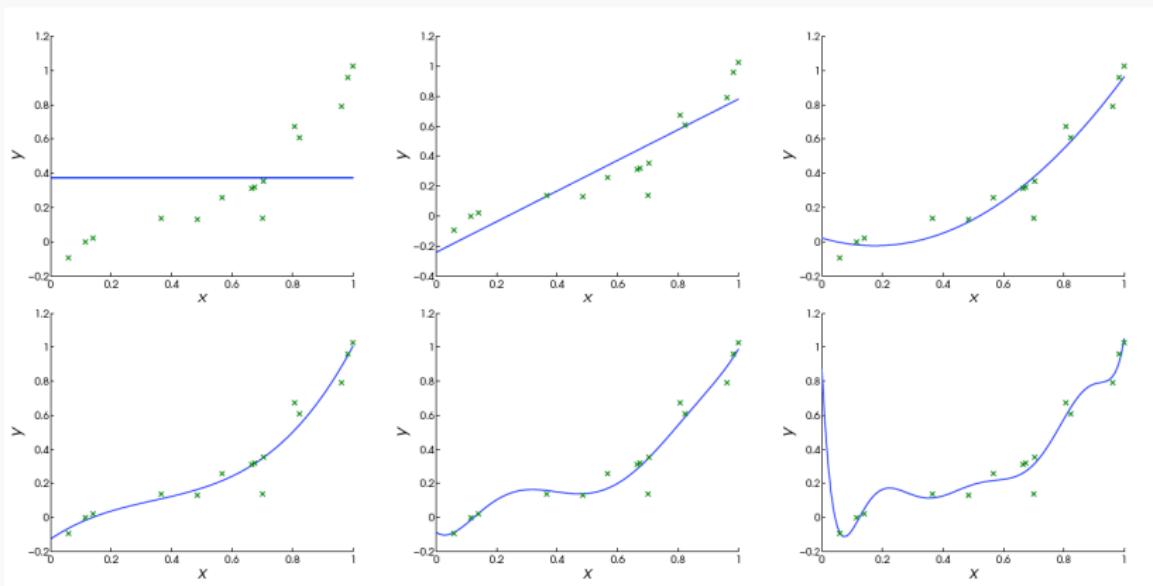
$$H_2 = \{f(x) = a_1x + a_2x^2 + b : a_1, a_2, b \in \mathbb{R}\}$$

⋮

$$H_n = \left\{ f(x) = \sum_{\ell=1}^n a_\ell x^\ell + b : a_1, \dots, a_n, b \in \mathbb{R} \right\}$$

Consider minimizing the empirical error in \mathcal{H}_r (r = “polynomial degree”)

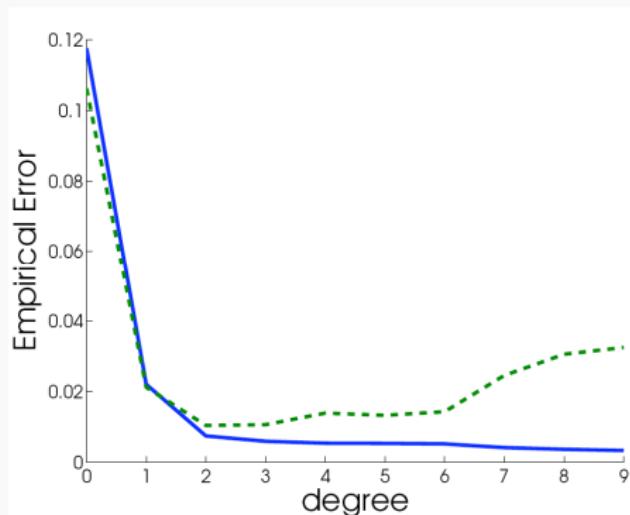
Polynomial fitting (simulation)



$r = 0, 1, 2, 3, 4, 5$. As r increases the fit to the data improves
(empirical error decreases)

Overfitting vs. Underfitting

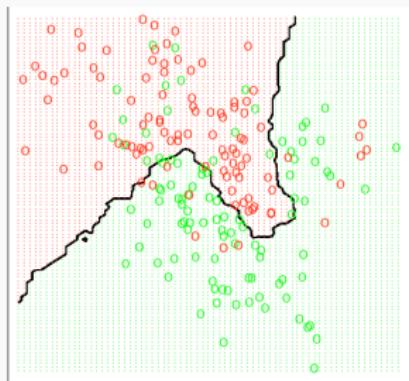
- Compare the empirical error (solid line) with expected error (dashed line)
 - r small: underfitting
 - r large: overfitting
- The larger r the lower the empirical error of f_S ! \Rightarrow We cannot rely on the training error!



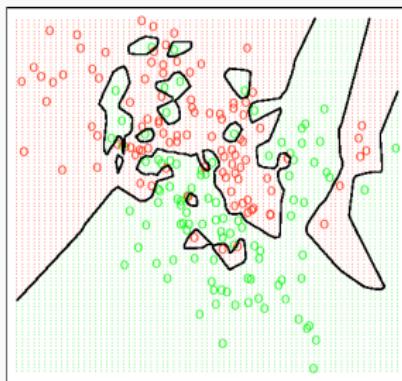
k -NN: the effect of k

- The smaller k the more irregular the decision boundary

$k = 15$



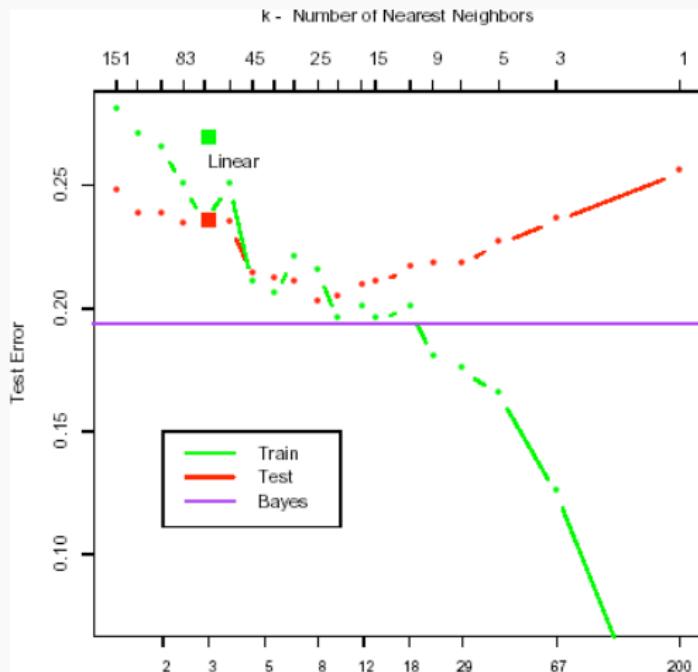
$k = 1$



- How to choose k ? later...

k -NN: the effect of k

$\frac{m}{k}$ large: overfitting versus $\frac{m}{k}$ small: underfitting



Model Selection

1. How to choose k in k -NN?
2. How to choose the degree r for polynomial regression?
3. The simplest approach is to use part of the training data (say 2/3) for training and the rest as a validation set for each \mathcal{H}_r
4. Another approach is K –fold cross-validation – see next slide
5. Then choose the “best” r relative to the error(s) on the validation set
6. We will return to model selection later in the course

Cross-validation

1. we split the data in K parts (of roughly equal sizes)
2. repeatedly train on $K - 1$ parts and test on the part “left out”
3. average the errors of K “validation” sets to give so-called cross-validation error
4. smaller K is less expensive but poorer estimate as size of training set is smaller and random fluctuations larger

For a dataset of size m , m -fold cross-validation is referred to as leave-one-out (LOO) testing

Cross-validation comments

- Cross validation is good in “practice.”
- There are a variety of theoretical-based approaches (not covered today)
- Examples
 1. “Bayesian” model selection via the “evidence”
 2. Structural Risk Minimization

Other learning paradigms

- **Online learning:** we observe the data sequentially and we make a prediction and update our learner after every datum.
- **Active learning:** we are given many inputs and we can choose which ones to request a label.
- **Unsupervised learning:** we have only input examples. Here we may want to find data clusters, estimate the probability density of the data, find important features/variable (dimensionality reduction problem), detect anomalies, etc.
- **Semi-supervised learning:** the ‘learning environment’ may give us access to many input examples but only few of them are labeled.
- **Reinforcement learning:** Similar to online learning where data is received sequentially but feedback is often delayed.

Suggested Readings

- Elements of Statistical Learning, 2ed, Chapter 2 : spans most of material of this lecture, with the following exceptions
- Cover & Hart : *Nearest Neighbor Pattern Classification*, 1967 : for Asymptotic Optimality of k -NN
- *Understanding Machine Learning from Theory to Algorithms*, Chapter 5.1 : For the NFL Theorem. See section 19.2.2 for a discussion of the curse of dimensionality wrt $K - nn$.

Going Deeper ...

- *Reconciling modern machine learning practice and the bias-variance trade-off.* Suggests that the classic U-shaped curve (see page 66) can be replaced by a double U, in a number of scenarios.
- *An adaptive nearest neighbor rule for classification.* Gives bounds for a k -NN variant that adapts k locally for each neighbourhood.

Problems – 1

1. True or False?

"For the K-nearest neighbour algorithm, larger K values will tend to lead to overfitting."

Explain your answer.

2.
 - First, give Cover's bound on the Bayes error of the 1-nearest neighbour algorithm as the number of examples goes to infinity.
 - Second, give an example where the 1-nearest neighbour algorithm achieves the Bayes error as the number of examples goes to infinity, explain your reasoning.
3. How can we choose k to ensure that the k -NN makes an unambiguous decision in the two-class case provided the test point is not equidistant to any pair of points in the training set? What values of k give an unambiguous decision under similar assumptions for the 3-class case?

Problems – 2

1. One of the drawbacks of the nearest-neighbour algorithm is that we must retain all of the training data. Describe a situation where a training point can be removed without affecting the resulting 1-NN classification for any test point in the input space.
2. In linear regression it is common to transform the training data

$$(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$$

to

$$((\mathbf{x}_1, 1), y_1), \dots, ((\mathbf{x}_m, 1), y_m)$$

i.e., we add an additional component to each input vector and set it to 1. What is the motivation for this procedure?

3. Given we have a dataset $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$ where $\mathbf{x} \in \mathbb{R}^n$ and $y \in \{1, 2, \dots, K\}$. Thus the inputs are real vectors and the labels denote one of K classes. How can linear regression (least squares), be used or adapted to tackle such a multi-class classification problem?

2. Kernels and Regularisation

COMP0078: Supervised Learning

Mark Herbster

11 October 2021

University College London
Department of Computer Science
SL-kernreg21v5

Today's Plan

Overview

- Inner product space review
- Convexity review
- Ridge Regression
- Basis Functions (Explicit Feature Maps)
- Kernel Functions (Implicit Feature Maps)

Overview

- We show how a linear method such as least squares may be **lifted** to a (potentially) higher dimensional space to provide a nonlinear regression.
- We consider both **explicit** and **implicit** feature maps
- A feature map is simply a function that maps the “inputs” into a new space.
- Thus the original method is now nonlinear in original “inputs” but linear in the “mapped inputs”
- Explicit feature maps are often known as the *Method of Basis Functions*
- Implicit feature maps are often known as the (*reproducing*) “Kernel Trick”

Review: Inner Product Space

Vector Space

Vector space over the reals

The triple $(X, +, *)$ defines a *vector space* where X is a set and $+ : X \times X \rightarrow X$ is vector addition and $* : \mathbb{R} \times X \rightarrow X$ is scalar multiplication with the abbreviation $a\mathbf{x} := a * \mathbf{x}$. For which the following properties hold.

1. $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$
2. $(\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z})$
3. There exists $\mathbf{0} \in X$ such that for all $\mathbf{x} \in X$ then $\mathbf{x} + \mathbf{0} = \mathbf{x}$
4. $\gamma(\mathbf{x} + \mathbf{y}) = \gamma\mathbf{x} + \gamma\mathbf{y}$
5. $(\gamma + \mu)\mathbf{x} = \gamma\mathbf{x} + \mu\mathbf{x}$
6. $\gamma(\mu\mathbf{x}) = (\gamma\mu)\mathbf{x}$
7. $0\mathbf{x} = \mathbf{0}$ and $1\mathbf{x} = \mathbf{x}$

Notes

1. $\gamma + \mu$ and $\gamma\mu$ are the usual scalar addition and multiplication over \mathbb{R}
2. A vector space can be defined over other *fields* than the reals for example arithmetic mod-2. I.e $X = \{\mathbf{0}, \mathbf{1}\}$ and the scalar set is just $\{0, 1\}$.

Normed Space

A function $\|\cdot\| : X \rightarrow \mathbb{R}$ is a *norm* on a vector space if

1. $\|\mathbf{x}\| = 0 \Leftrightarrow \mathbf{x} = \mathbf{0}$
2. $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$
3. $\|\gamma \mathbf{x}\| = |\gamma| \|\mathbf{x}\|$

A norm defines a *metric* (distance) between vectors in the space via $d(\mathbf{x}, \mathbf{y}) := \|\mathbf{x} - \mathbf{y}\|$. Check that the triangle inequality holds
 $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$.

Examples

1. $\|\mathbf{x}\|_p := (\sum_{i=1}^n |x_i^p|)^{\frac{1}{p}}$ where $\mathbf{x} \in \mathbb{R}^n$ and $p \in [1, \infty)$.
2. $\|\mathbf{x}\|_M := \sqrt{\mathbf{x}^\top M \mathbf{x}}$ where $\mathbf{x} \in \mathbb{R}^n$ and M is an $n \times n$ symmetric positive definite matrix.

Definition

A real-valued symmetric $n \times n$ square matrix M is *positive definite* iff $\mathbf{x}^\top M \mathbf{x} > 0$ ($\forall \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$).

Normed Space – Intuitions

A norm generalises the notion of euclidean magnitude $\|\cdot\|_2$. The p -norm plays a particularly important role in machine learning where kernel methods may be seen as a generalisation of the case $p = 2$. The case $p = 1$ is particularly important when learning sparse predictors.

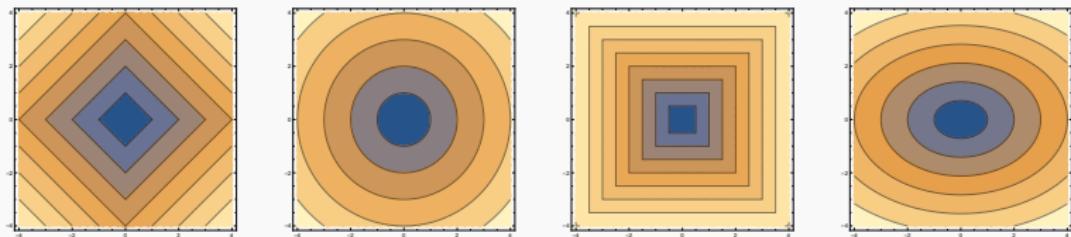


Figure 1: Level sets of $\|\cdot\|_1$, $\|\cdot\|_2$, $\|\cdot\|_\infty$, $\|\cdot\|_{(1,0), (0,3)}$

Where $\|\mathbf{x}\|_\infty := \lim_{p \rightarrow \infty} \|\mathbf{x}\|_p = \max_{i \in [n]} x_i$.

Level set of f at α is $\{x : f(x) \leq \alpha\}$.

Question : What metric does $\|\cdot\|_1$ correspond to in \mathbb{R}^2 ?

Real Inner product space

The quadruple $(X, +, *, \langle \cdot, \cdot \rangle)$ defines an *inner product space* where $(X, +, *)$ is a vector space and $\langle \cdot, \cdot \rangle : X \times X \rightarrow \mathbb{R}$ is an inner product s.t.

1. $\langle \mathbf{x}, \mathbf{x} \rangle = 0 \Leftrightarrow \mathbf{x} = \mathbf{0}$ and $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$
2. $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$
3. $\langle \gamma \mathbf{x}, \mathbf{y} \rangle = \gamma \langle \mathbf{x}, \mathbf{y} \rangle$ and $\langle \mathbf{x} + \mathbf{y}, \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{z} \rangle + \langle \mathbf{y}, \mathbf{z} \rangle$

The inner product induces a norm via $\|\mathbf{x}\| := \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$. The geometric interpretation of the inner product so that if $\theta := \cos^{-1}\left(\frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|}\right)$ then θ is the angle between the vectors.

Examples

1. The Euclidean inner product $\langle \mathbf{x}, \mathbf{y} \rangle := \sum_{i=1}^n x_i y_i$
2. More generally $\langle \mathbf{x}, \mathbf{y} \rangle_M := \mathbf{x}^\top M \mathbf{y}$ where $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and M is an $n \times n$ symmetric positive definite matrix. (**Exercise: Check**)

Note: Many different notations are used for inner product including
 $\langle \mathbf{x}, \mathbf{y} \rangle = (\mathbf{x}, \mathbf{y}) = \mathbf{x} \cdot \mathbf{y}$

A more complicated example – 1

Here the space is set of all functions from $[0, \infty) \rightarrow \mathbb{R}$ with the following three properties.

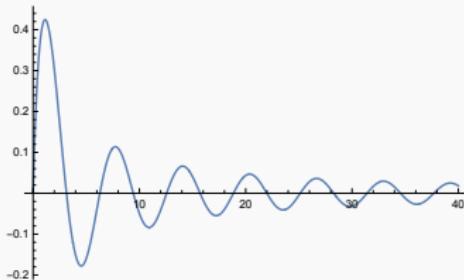
1. $f(0) = 0$
2. f is absolutely continuous (hence $f(b) - f(a) = \int_a^b f'(x)dx$)
3. $\int_0^\infty [f'(x)]^2 dx < \infty$

Addition ‘+’ is the usual addition of functions similarly multiplication by a scalar. The inner product is defined as

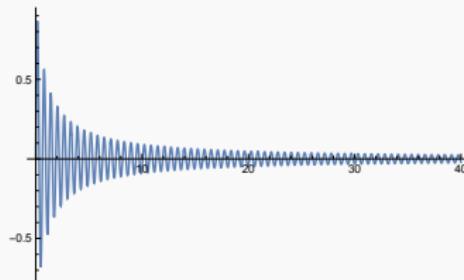
$$\langle f, g \rangle := \int_0^\infty f'(x)g'(x)dx$$

Exercise : Argue that this is an inner product space.

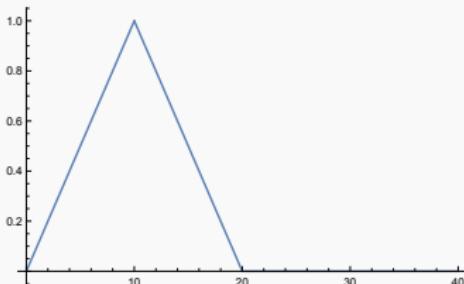
A more complicated example – 2



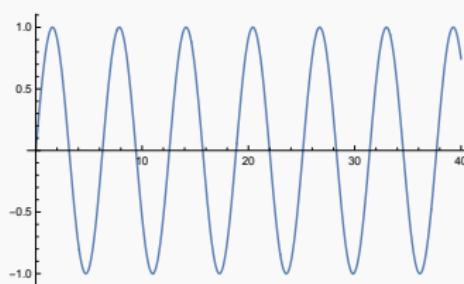
(a) $f(x) = \frac{\sin(x)}{x+1}; \|f\| \approx 0.68$



(b) $f(x) = \frac{\sin(10x)}{x+1}; \|f\| \approx 7.06$



(c) $f(x) = .1x[x \leq 10] + (2 - .1x)[10 < x \leq 20]; \|f\| = \frac{1}{\sqrt{5}}$



(d) $f(x) = \sin(x); \text{Not in space (why?)}$

Figure 2: Example functions and their norms

Review: Convexity

Convexity

Definition

A function $f : \mathcal{X} \rightarrow \mathbb{R}$ is **convex** iff $\forall p, q \in \mathcal{X}$ and $\alpha \in (0, 1)$ we have

$$f(\alpha p + (1 - \alpha)q) \leq \alpha f(p) + (1 - \alpha)f(q)$$

A function f is **concave** if $-f$ is convex. A function is additionally **strictly convex** if we can replace " \leq " with " $<$ " when $p \neq q$.

Definition

A set \mathcal{X} is convex if $p, q \in \mathcal{X} \implies (\alpha p + (1 - \alpha)q) \in \mathcal{X}$ for $\forall \alpha \in (0, 1)$.

Some results

1. If f and g are convex then $f + g$ is convex.
2. If f is convex and g is affine (linear + a constant) then $f(g(\cdot))$ is convex.
3. Suppose M is a symmetric matrix then M is a PSD matrix iff
 $f(x) = x^\top Mx$ is convex.
4. A level set of a convex function is convex.

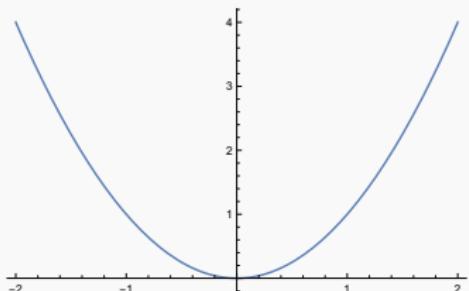
Exercise: prove the results above.

Checking for convexity

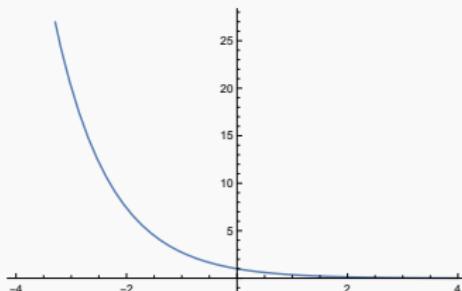
Some Results

1. For $f : (a, b) \rightarrow \mathbb{R}$ if f' is increasing then f is convex.
2. For $f : (a, b) \rightarrow \mathbb{R}$ if $f'' \geq 0$ then f is convex.
3. For $f : \mathcal{X} \rightarrow \mathbb{R}$ if $\mathcal{X} \subseteq \mathbb{R}^n$, \mathcal{X} is a convex set and if the Hessian of f evaluated at x denoted $H(x)$ is positive semidefinite for all $x \in \mathcal{X}$ then f is convex.

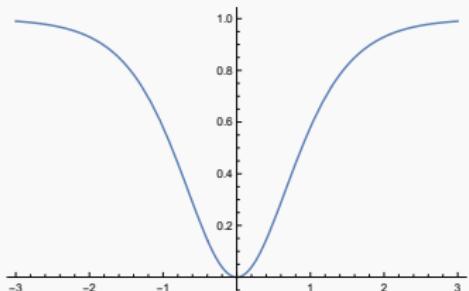
Examples



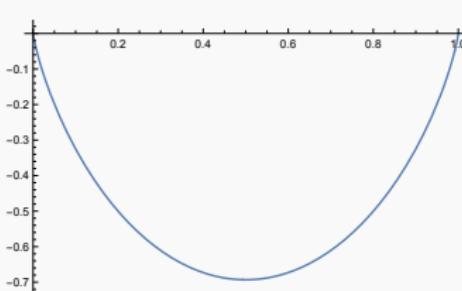
(a) $f(x) = x^2$ "Quadratic"



(b) $f(x) = e^{-x}$ "Exponential decay"



(c) $f(x) = \tanh(x)^2$ "Neural" (not convex)



(d) $f(x) = x \log(x) + (1 - x) \log(1 - x)$; "negative entropy"

Why Convexity?

- At the heart of many ML algorithms is an optimisation problem.
- For example, minimize error, minimise regularised error, minimise “energy”, maximise likelihood.
- If the (unconstrained) optimisation prob. is convex and there is a minima¹ then methods gradient-based methods can be applied to smooth problems.
- On the other hand. The existence of “many” minima each associated with a distinct function suggests that for many optimisation approaches there will be a high “variance”.
- Take-away : everything else equal convex objectives in ML are simpler to work with.

¹Or more technically a minimal surface.

Ridge Regression

Linear interpolation

Problem

We wish to find a function $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ which best interpolates a data set $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\} \subseteq \mathbb{R}^n \times \mathbb{R}$

- If the data have been generated in the form $(\mathbf{x}, f(\mathbf{x}))$, the vectors \mathbf{x}_i are linearly independent and $m = n$ then there is a unique interpolant whose parameter \mathbf{w} solves

$$X\mathbf{w} = \mathbf{y}$$

where, recall, $\mathbf{y} = (y_1, \dots, y_m)^\top$ and $X = [\mathbf{x}_1, \dots, \mathbf{x}_m]^\top$

- Otherwise, this problem is *ill-posed*

III-posed problems

A problem is well-posed – in the sense of Hadamard (1902) – if

- (1) a solution exists
- (2) the solution is unique
- (3) the solution depends continuously on the data

A problem is ill-posed if it is not well-posed

Learning problems are in general ill-posed (usually because of (2))

Regularization theory provides a general framework to solve ill-posed problems

Ridge Regression

Motivation:

1. Give a set of k hypothesis classes $\{\mathcal{H}_r\}_{r \in \mathbb{N}_k}$ we can choose an appropriate hypothesis class with *cross-validation*
2. An alternative compatible with linear regression is to choose a single “complex” hypothesis class and then modify the error function by adding a “complexity” term which penalizes complex functions
3. This is known as **regularization**
4. Cross-validation may still be needed to set the regularization parameter (see below) and other parameters defining the complexity term

Ridge Regression

We minimize the regularized (penalized) empirical error

$$\mathcal{E}_{\text{emp}}_{\lambda}(\mathbf{w}) := \sum_{i=1}^m (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \lambda \sum_{\ell=1}^n w_\ell^2 \equiv (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2$$

The positive parameter λ defines a trade-off between the error on the data and the norm of the vector \mathbf{w} (degree of regularization)

Setting $\nabla \mathcal{E}_{\text{emp}}_{\lambda}(\mathbf{w}) = 0$, we obtain the modified normal equations

$$-\mathbf{2X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda\mathbf{w} = 0 \tag{1}$$

whose solution (called *regularized solution*) is

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X} + \lambda I_n)^{-1} \mathbf{X}^\top \mathbf{y} \tag{2}$$

Dual representation

It can be shown that the regularized solution can be written as

$$\mathbf{w} = \sum_{i=1}^m \alpha_i \mathbf{x}_i \quad \Rightarrow \quad f(\mathbf{x}) = \sum_{i=1}^m \alpha_i \mathbf{x}_i^\top \mathbf{x} \quad (*)$$

where the vector of parameters $\alpha = (\alpha_1, \dots, \alpha_m)^\top$ is given by

$$\alpha = (X X^\top + \lambda I_m)^{-1} \mathbf{y} \quad (3)$$

- **Function representations:** we call the functional form (or representation) $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ the *primal form* and (*) the *dual form* (or representation)

The dual form is computationally convenient when $n > m$

Dual representation (continued – 1)

We rewrite eq.(1) as

$$\mathbf{w} = \frac{\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w})}{\lambda}$$

Thus we have

$$\mathbf{w} = \sum_{i=1}^m \alpha_i \mathbf{x}_i \quad (4)$$

with

$$\alpha_i = \frac{y_i - \mathbf{w}^\top \mathbf{x}_i}{\lambda} \quad (5)$$

Consequently, we have that

$$\mathbf{w}^\top \mathbf{x} = \sum_{i=1}^m \alpha_i \mathbf{x}_i^\top \mathbf{x}$$

proving eq.(*)

Dual representation (continued – 2)

Plugging eq.(4) in eq.(5) we obtain

$$\alpha_i = \frac{y_i - (\sum_{j=1}^m \alpha_j \mathbf{x}_j)^\top \mathbf{x}_i}{\lambda}$$

Thus (with defining $\delta_{ij} = 1$ if $i = j$ and as 0 otherwise)

$$y_i = (\sum_{j=1}^m \alpha_j \mathbf{x}_j)^\top \mathbf{x}_i + \lambda \alpha_i$$

$$y_i = \sum_{j=1}^m (\alpha_j \mathbf{x}_j^\top \mathbf{x}_i + \alpha_j \lambda \delta_{ij})$$

$$y_i = \sum_{j=1}^m (\mathbf{x}_j^\top \mathbf{x}_i + \lambda \delta_{ij}) \alpha_j$$

Hence $(XX^\top + \lambda I_m)\alpha = \mathbf{y}$ from which eq.(3) follows.

Computational Considerations

Training time:

- Solving for \mathbf{w} in the primal form requires $O(mn^2 + n^3)$ operations while solving for α in the dual form requires $O(nm^2 + m^3)$ (see $(*)$) operations

If $m \ll n$ it is more efficient to use the dual representation

Running (testing) time:

- Computing $f(\mathbf{x})$ on a test vector \mathbf{x} in the primal form requires $O(n)$ operations while the dual form (see $(*)$) requires $O(mn)$ operations

Sparse representation

We can benefit even further in the dual representation if the inputs are sparse!

Example

Suppose each input $\mathbf{x} \in \mathbb{R}^n$ has most of its components equal to zero (e.g., consider images where most pixels are ‘black’ or text documents represented as ‘bag of words’)

- If k denotes the number of nonzero components of the input then computing $\mathbf{x}^\top \mathbf{t}$ requires at most $O(k)$ operations.
How do we do this?
- If $km \ll n$ (which implies $m, k \ll n$) the dual representation requires $O(km^2 + m^3)$ computations for training and $O(mk)$ for testing

Basis Functions

Basis Functions – Explicit Feature Map

The above ideas can naturally be generalized to nonlinear function regression

By a *feature map* we mean a function $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^N$

$$\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_N(\mathbf{x}))^\top, \quad \mathbf{x} \in \mathbb{R}^n$$

- The ϕ_1, \dots, ϕ_N are called basis functions
- Vector $\phi(\mathbf{x})$ is called the *feature vector* and the space

$$\{\phi(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^n\}$$

the *feature space*

The non-linear regression function has the primal representation

$$f(\mathbf{x}) = \sum_{j=1}^N w_j \phi_j(\mathbf{x})$$

Feature Maps (Example 1 : [BIAS])

We've already seen one example with $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^{n+1}$

$$\phi(\mathbf{x}) = (\mathbf{x}, 1)^\top$$

In the context of linear regression before application of the feature map we had

$$\arg \min_{\mathbf{w} \in \mathbb{R}^n} \sum_{i=1}^m (\mathbf{w}^\top \mathbf{x}_i - y)^2$$

After the feature map we have

$$\arg \min_{\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}} \sum_{i=1}^m (\mathbf{w}^\top \mathbf{x}_i + b - y_i)^2$$

thus allowing us to learn a linear fit with a constant offset.

Feature Maps (Example 2 : [XOR])

Consider the XOR function defined as

x_1	x_2	$x_1 \text{ XOR } x_2$
1	1	-1
1	-1	1
-1	1	1
-1	-1	-1

Does there exist a linear classifier that fits XOR perfectly? What if we add a bias term? **Why?**

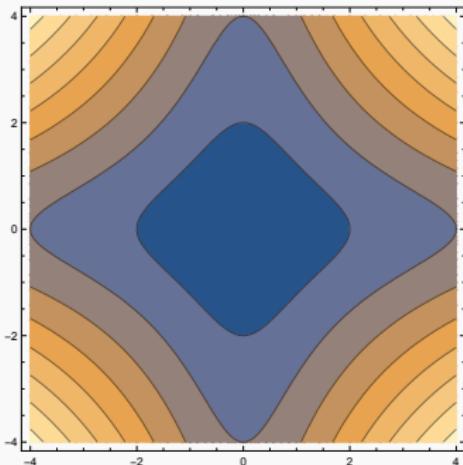
What if instead we first apply the feature map $\phi(\mathbf{x}) := (\mathbf{x}, x_1 x_2)^\top$?

Feature Maps (Example 2 : [XOR] – continued)

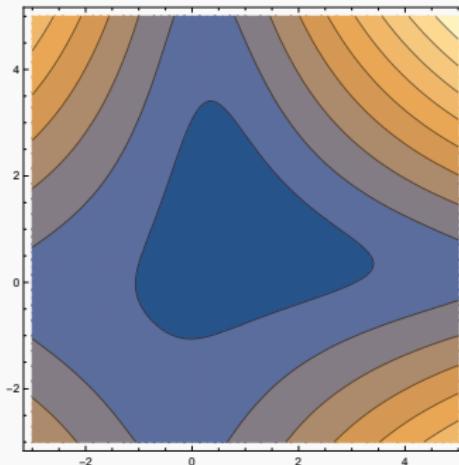
Let's visualize the unit balls associated with the “correlation” feature map

The distance between two points \mathbf{x}, \mathbf{t} after mapping in feature space is

$$\|\phi(\mathbf{x}) - \phi(\mathbf{t})\|_2 = \sqrt{(x_1 - t_1)^2 + (x_2 - t_2)^2 + (x_1 x_2 - t_1 t_2)^2}$$



(a) Ball centred at $(0, 0)$.



(b) Ball centred at $(1, 1)$.

Figure 3: Unit ball of 2d correlation feature map in “original space”

- Observe that the unit balls of in R^2 w.r.t. to the feature-mapped

Feature (Example 3 : Correlations)

More generally the trick behind XOR was to add to the original vector the “correlate” x_1x_2 .

More generally for second order correlations if $\mathbf{x} \in \mathbb{R}^n$ we have

$$\phi(\mathbf{x}) := (\mathbf{x}, x_1x_1, x_1x_2, \dots, x_1x_n, x_2x_2, x_2x_3, \dots, x_2x_n, \dots, x_nx_n)^\top$$

i.e., $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^{\frac{n^2+3n}{2}}$.

What is the motivation for this feature map?

More generally we might also include higher order correlations.

What is a potential problem with this technique?

Kernels

Computational Considerations Revisited

Again, if $m \ll N$ it is more efficient to work with the dual representation

Key observation: in the dual representation we don't need to know ϕ explicitly; we just need to know the inner product between any pair of feature vectors!

Example: Consider the following feature map with second order correlations ($N = n^2$)

$$\phi(\mathbf{x}) = (x_1x_1, x_1x_2, \dots, x_nx_n)^\top$$

$$\begin{aligned}\langle \phi(\mathbf{x}), \phi(\mathbf{t}) \rangle &= (x_1x_1, x_1x_2, \dots, x_nx_n)(t_1t_1, t_1t_2, \dots, t_nt_n)^\top \\ &= x_1x_1t_1t_1 + x_1x_2t_1t_2 + \dots + x_nx_nt_nt_n \\ &= (x_1t_1 + \dots + x_nt_n)(x_1t_1 + \dots + x_nt_n) \\ &= (\mathbf{x}^\top \mathbf{t})^2\end{aligned}$$

Observe that $(\mathbf{x}^\top \mathbf{t})^2$ requires only $O(n)$ computations whereas the more direct $(x_1x_1, x_1x_2, \dots, x_nx_n)(t_1t_1, t_1t_2, \dots, t_nt_n)^\top$ requires $O(n^2)$

Kernel Functions – Implicit Feature Map

Given a feature map ϕ we define its associated kernel function $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ as

$$K(\mathbf{x}, \mathbf{t}) := \langle \phi(\mathbf{x}), \phi(\mathbf{t}) \rangle, \quad \mathbf{x}, \mathbf{t} \in \mathbb{R}^n$$

- **Key Point:** for some feature map ϕ computing $K(\mathbf{x}, \mathbf{t})$ is independent of N (only dependent on n). Where *necessarily* $\phi(\mathbf{x})$ depends on N .

Example (cont.) If $\phi(\mathbf{x}) = (x_{i_1} x_{i_2} \cdots x_{i_r} : i_1, \dots, i_r \in \{1, \dots, n\})$ then we have that

$$K(\mathbf{x}, \mathbf{t}) = (\mathbf{x}^\top \mathbf{t})^r$$

In this case $K(\mathbf{x}, \mathbf{t})$ is computed with $O(n)$ operations, which is essentially independent of r or $N = n^r$. On the other hand, computing $\phi(\mathbf{x})$ requires $O(N)$ operations – **Exponential in $r!$** .

Question: So far the feature map has all r-order correlates how can we change it so that (r-1)-order, (r-2)-order, etc., correlates are

Redundancy of the feature map

Warning

The feature map is not unique! If ϕ generates K so does $\hat{\phi} = \mathbf{U}\phi$ where \mathbf{U} is an (any!) $N \times N$ orthogonal matrix. Even the dimension of ϕ is not unique!

Proof.

$$(\mathbf{U}\phi)^\top (\mathbf{U}\phi) = \phi^\top \mathbf{U}^\top \mathbf{U}\phi = \phi^\top \phi$$

Example

If $n = 2$, $K(\mathbf{x}, \mathbf{t}) = (\mathbf{x}^\top \mathbf{t})^2$ is generated by both
 $\phi(\mathbf{x}) = (x_1^2, x_2^2, x_1 x_2, x_2 x_1)$ and $\hat{\phi}(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1 x_2)$.

Regularization-based learning algorithms

Let us open a short parenthesis and show that the dual form of ridge regression holds true for other loss functions as well

$$\mathcal{E}_{\text{emp}}_{\lambda}(\mathbf{w}) = \sum_{i=1}^m V(y_i, \langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle) + \lambda \langle \mathbf{w}, \mathbf{w} \rangle, \quad \lambda > 0 \quad (6)$$

where $V : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is a loss function

Theorem

If V is differentiable wrt. its second argument and \mathbf{w} is a minimizer of E_{λ} then it has the form

$$\mathbf{w} = \sum_{i=1}^m \alpha_i \phi(\mathbf{x}_i) \Rightarrow f(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle = \sum_{i=1}^m \alpha_i K(\mathbf{x}_i, \mathbf{x})$$

This result is usually called the *Representer Theorem*

Representer theorem

Setting the derivative of E_λ wrt. \mathbf{w} to zero we have

$$\sum_{i=1}^m V'(y_i, \langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle) \phi(\mathbf{x}_i) + 2\lambda \mathbf{w} = \mathbf{0} \Rightarrow \mathbf{w} = \sum_{i=1}^m \alpha_i \phi(\mathbf{x}_i) \quad (7)$$

where V' is the partial derivative of V wrt. its second argument and we defined

$$\alpha_i = \frac{1}{2\lambda} V'(y_i, \langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle) \quad (8)$$

Thus we conclude that

$$f(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle = \sum_{i=1}^m \alpha_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle = \sum_{i=1}^m \alpha_i K(\mathbf{x}, \mathbf{x}_i),$$

Some remarks

- Plugging eq.(7) in the rhs. of eq.(8) we obtain a set of equations for the coefficients α_i :

$$\alpha_i = \frac{1}{2\lambda} V' \left(y_i, \sum_{j=1}^m K(\mathbf{x}_i, \mathbf{x}_j) \alpha_j \right), \quad i = 1, \dots, m$$

When V is the square loss and $\phi(\mathbf{x}) = \mathbf{x}$ we retrieve the linear eq.(5)

- Substituting eq.(7) in eq.(6) we obtain an objective function for the α 's:

$$\sum_{i=1}^m V(y_i, (\mathbf{K}\boldsymbol{\alpha})_i) + \lambda \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha}, \quad \text{where : } \mathbf{K} = (K(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1}^m$$

Remark: the Representer Theorem holds true under more general conditions on V (for example V can be any continuous function)

What functions are “kernels”?

Question

Given a function $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$, which properties of K guarantee that there exists a *Hilbert space* \mathcal{H} and a feature map $\phi : \mathbb{R}^n \rightarrow \mathcal{H}$ such that $K(\mathbf{x}, \mathbf{t}) = \langle \phi(\mathbf{x}), \phi(\mathbf{t}) \rangle$?

Note 1

We've generalized the definition of *finite-dimensional* feature maps

$$\phi : \mathbb{R}^n \rightarrow \mathbb{R}^N$$

to now allow potentially *infinite-dimensional* feature maps

$$\phi : \mathbb{R}^n \rightarrow \mathcal{H}$$

Note (technical) 2

A Hilbert space is an inner product space which also contains the limit points of all its Cauchy sequences.

Positive Semidefinite Kernel

Definition

A function $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ is **positive semidefinite** if it is symmetric and the matrix $(K(\mathbf{x}_i, \mathbf{x}_j) : i, j = 1, \dots, k)$ is positive semidefinite for every $k \in \mathbb{N}$ and every $\mathbf{x}_1, \dots, \mathbf{x}_k \in \mathbb{R}^n$

Theorem

K is positive semidefinite if and only if

$$K(\mathbf{x}, \mathbf{t}) = \langle \phi(\mathbf{x}), \phi(\mathbf{t}) \rangle, \quad \mathbf{x}, \mathbf{t} \in \mathbb{R}^n$$

for some feature map $\phi : \mathbb{R}^n \rightarrow \mathcal{W}$ and Hilbert space \mathcal{W}

Note. We may replace domain \mathbb{R}^n by any abstract set \mathcal{X} in the above definitions.

Positive semidefinite kernel (cont.)

Proof of " \Leftarrow "

If $K(\mathbf{x}, \mathbf{t}) = \langle \phi(\mathbf{x}), \phi(\mathbf{t}) \rangle$ then we have that

$$\sum_{i,j=1}^m c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) = \left\langle \sum_{i=1}^m c_i \phi(\mathbf{x}_i), \sum_{j=1}^m c_j \phi(\mathbf{x}_j) \right\rangle = \left\| \sum_{i=1}^m c_i \phi(\mathbf{x}_i) \right\|^2 \geq 0$$

for every choice of $m \in \mathbb{N}$, $x_i \in \mathbb{R}^d$ and $c_i \in R$, $i = 1, \dots, m$

Note

the proof of ' \Rightarrow ' requires the notion of reproducing kernel Hilbert spaces.

Informally, one can show that the linear span of the set of functions

$\{K(\mathbf{x}, \cdot) : \mathbf{x} \in \mathbb{R}^n\}$ can be made into a Hilbert space H_K with inner product induced by the definition $\langle K(\mathbf{x}, \cdot), K(\mathbf{t}, \cdot) \rangle_K := K(\mathbf{x}, \mathbf{t})$. In particular, the map $\phi : \mathbb{R}^n \rightarrow H_K$ defined as $\phi(\mathbf{x}) = K(\mathbf{x}, \cdot)$ is a feature map associated with K .

Observe (check!) then with $f(\cdot) := \sum_{i=1}^m \alpha_i K(\mathbf{x}_i, \cdot)$ that

$$\|f\|^2 = \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j).$$

Two Example Kernels

Polynomial Kernel(s)

If $p : \mathbb{R} \rightarrow \mathbb{R}$ is a polynomial with nonnegative coefficients then $K(\mathbf{x}, \mathbf{t}) = p(\mathbf{x}^\top \mathbf{t})$, $\mathbf{x}, \mathbf{t} \in \mathbb{R}^n$ is a positive semidefinite kernel. For example if $a \geq 0$

- $K(\mathbf{x}, \mathbf{t}) = (\mathbf{x}^\top \mathbf{t})^r$
- $K(\mathbf{x}, \mathbf{t}) = (a + \mathbf{x}^\top \mathbf{t})^r$
- $K(\mathbf{x}, \mathbf{t}) = \sum_{i=0}^d \frac{a^i}{i!} (\mathbf{x}^\top \mathbf{t})^i$

are each positive semidefinite kernels.

Gaussian Kernel

An important example of a “radial” kernel is the Gaussian kernel

$$K(\mathbf{x}, \mathbf{t}) = \exp(-\beta \|\mathbf{x} - \mathbf{t}\|^2), \quad \beta > 0, \mathbf{x}, \mathbf{t} \in \mathbb{R}^n$$

note: any corresponding feature map $\phi(\cdot)$ is ∞ -dimensional.

Polynomial and Anova Kernel

Anova Kernel

$$K_a(\mathbf{x}, \mathbf{t}) = \prod_{i=1}^n (1 + x_i t_i)$$

Compare to the polynomial kernel $K_p(\mathbf{x}, \mathbf{t}) = (1 + \mathbf{x}^\top \mathbf{t})^d$

where $\sum_{i=0}^n i_j = d$

Problem: Argue that $\langle \phi_a(\mathbf{x}), \phi_a(\mathbf{t}) \rangle = K_a(\mathbf{x}, \mathbf{t})$.

Kernel construction

Which operations/combinations (eg, products, sums, composition, etc.) of a given set of kernels is still a kernel?

If we address this question we can build more interesting kernels starting from simple ones

Example

We have already seen that $K(\mathbf{x}, \mathbf{t}) = (\mathbf{x}^\top \mathbf{t})^r$ is a kernel. For which class of functions $p : \mathbb{R} \rightarrow \mathbb{R}$ is $p(\mathbf{x}^\top \mathbf{t})$ a kernel? More generally, if K is a kernel when is $p(K(\mathbf{x}, \mathbf{t}))$ a kernel?

General linear kernel

If \mathbf{A} is an $n \times n$ psd matrix the function $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ defined by

$$K(\mathbf{x}, \mathbf{t}) = \mathbf{x}^\top \mathbf{A} \mathbf{t}$$

is a kernel

Proof

Since \mathbf{A} is psd we can write it in the form $\mathbf{A} = \mathbf{R}\mathbf{R}^\top$ for some $n \times n$ matrix \mathbf{R} . Thus K is represented by the feature map $\phi(\mathbf{x}) = \mathbf{R}^\top \mathbf{x}$

Alternatively, note that:

$$\begin{aligned} \sum_{i,j} c_i c_j \mathbf{x}_i^\top \mathbf{A} \mathbf{x}_j &= \sum_{i,j} c_i c_j (\mathbf{R}^\top \mathbf{x}_i)^\top (\mathbf{R}^\top \mathbf{x}_j) = \\ &\quad \sum_i c_i [(\mathbf{R}^\top \mathbf{x}_i)]^\top [\sum_j c_j (\mathbf{R}^\top \mathbf{x}_j)] = \left\| \sum_i c_i \mathbf{R}^\top \mathbf{x}_i \right\|^2 \geq 0 \end{aligned}$$

Kernel composition

More generally, if $K : \mathbb{R}^N \times \mathbb{R}^N \rightarrow \mathbb{R}$ is a kernel and $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^N$, then

$$\tilde{K}(\mathbf{x}, \mathbf{t}) = K(\phi(\mathbf{x}), \phi(\mathbf{t}))$$

is a kernel from $\mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$.

Proof

By hypothesis, K is a kernel and so, for every $\mathbf{x}_1, \dots, \mathbf{x}_m \in \mathbb{R}^n$ the matrix $(K(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)) : i, j = 1, \dots, m)$ is psd

In particular, the previous example corresponds to $K(\mathbf{x}, \mathbf{t}) = \mathbf{x}^\top \mathbf{t}$ and $\phi(\mathbf{x}) = \mathbf{R}^\top \mathbf{x}$

Kernel construction (cont.)

Question

If K_1, \dots, K_q are kernels on \mathbb{R}^n and $F : \mathbb{R}^q \rightarrow \mathbb{R}$, when is the function

$$F(K_1(\mathbf{x}, \mathbf{t}), \dots, K_q(\mathbf{x}, \mathbf{t})), \quad \mathbf{x}, \mathbf{t} \in \mathbb{R}^n$$

a kernel?

Equivalently: when for every choice of $m \in \mathbb{N}$ and $\mathbf{A}_1, \dots, \mathbf{A}_q$ $m \times m$ psd matrices, is the following matrix psd?

$$(F(A_{1,ij}, \dots, A_{q,ij}) : i, j = 1, \dots, m)$$

We discuss some examples of functions F for which the answer to these question is YES

Nonnegative combination of kernels

If $\lambda_j \geq 0$, $j = 1, \dots, q$ then $\sum_{j=1}^q \lambda_j K_j$ is a kernel

This fact is immediate (a non-negative combination of psd matrices is still psd)

Example: Let $q = n$ and $K_j(\mathbf{x}, \mathbf{t}) = x_j t_j$.

In particular, this implies that

- aK_1 is a kernel if $a \geq 0$
- $K_1 + K_2$ is a kernel

Product of kernels

The pointwise product of two kernels K_1 and K_2

$$K(\mathbf{x}, \mathbf{t}) := K_1(\mathbf{x}, \mathbf{t})K_2(\mathbf{x}, \mathbf{t}), \quad \mathbf{x}, \mathbf{t} \in \mathbb{R}^d$$

is a kernel

Proof

Idea: The fact that the element-wise product of PSD matrices is again PSD implies that product of kernels is again a kernel.

Thus we need to show that if \mathbf{A} and \mathbf{B} are psd matrices, so is

$\mathbf{C} = (A_{ij}B_{jj} : i, j = 1, \dots, n)$ (\mathbf{C} is also called the Schur product of \mathbf{A} and \mathbf{B}).

Since \mathbf{A} and \mathbf{B} are psd we can write them in the form $\mathbf{A} = \mathbf{U}\mathbf{U}^\top$ and

$\mathbf{B} = \mathbf{V}\mathbf{V}^\top$ for some $n \times n$ matrices \mathbf{U} and \mathbf{V} .

$$\begin{aligned} \sum_{i,j=1}^m z_i z_j C_{ij} &= \sum_{ij} z_i z_j \sum_r U_{ir} U_{jr} \sum_s V_{is} V_{js} = \sum_{ij} \sum_{rs} z_i z_j U_{ir} U_{jr} V_{is} V_{js} \\ &= \sum_{rs} \sum_{ij} z_i z_j U_{ir} U_{jr} V_{is} V_{js} = \sum_{rs} \sum_i z_i U_{ir} V_{is} \sum_j z_j U_{jr} V_{js} \\ &= \sum_{rs} \left(\sum_i z_i U_{ir} V_{is} \right)^2 \geq 0 \end{aligned}$$

Summary of constructions

Theorem

If K_1, K_2 are kernels, $a \geq 0$, A is a symmetric positive semi-definite matrix, K a kernel on \mathbb{R}^N and $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^N$ then the following functions are positive semidefinite kernels on \mathbb{R}^n

1. $\mathbf{x}^\top A \mathbf{t}$
2. $K_1(\mathbf{x}, \mathbf{t}) + K_2(\mathbf{x}, \mathbf{t})$
3. $aK_1(\mathbf{x}, \mathbf{t})$
4. $K_1(\mathbf{x}, \mathbf{t})K_2(\mathbf{x}, \mathbf{t})$
5. $K(\phi(\mathbf{x}), \phi(\mathbf{t}))$

Polynomial of kernels

Let $F = p$ where $p : \mathbb{R}^q \rightarrow \mathbb{R}$ is a polynomial in q variables with nonnegative coefficients. By properties 2,3 and 4 above we conclude that p is a valid function

In particular if $q = 1$,

$$\sum_{i=1}^d a_i (K(\mathbf{x}, \mathbf{t}))^i$$

is a kernel if $a_1, \dots, a_d \geq 0$

Polynomial kernels

The above observation implies that if $p : \mathbb{R} \rightarrow \mathbb{R}$ is a polynomial with nonnegative coefficients then $p(\mathbf{x}^\top \mathbf{t})$, $\mathbf{x}, \mathbf{t} \in \mathbb{R}^n$ is a kernel on \mathbb{R}^n . In particular if $a \geq 0$ the following are valid polynomial kernels

- $(\mathbf{x}^\top \mathbf{t})^r$
- $(a + \mathbf{x}^\top \mathbf{t})^r$
- $\sum_{i=0}^d \frac{a^i}{i!} (\mathbf{x}^\top \mathbf{t})^i$

'Infinite polynomial' kernel

If in the last equation we set $r = \infty$ the series

$$\sum_{i=0}^r \frac{a^i}{i!} (\mathbf{x}^\top \mathbf{t})^i$$

converges everywhere uniformly to $\exp(a\mathbf{x}^\top \mathbf{t})$ showing that this function is also a kernel.

Assume for simplicity that $n = 1$. A feature map corresponding to the kernel $\exp(axt)$ is

$$\phi(x) = \left(1, \sqrt{a}x, \sqrt{\frac{a}{2}}x^2, \sqrt{\frac{a^3}{6}}x^3, \dots \right) = \left(\sqrt{\frac{a^i}{i!}}x^i : i \in \mathbb{N} \right)$$

- The feature space has an infinite dimensionality!

Translation invariant and radial kernels

We say that a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is

- *Translation invariant* if it has the form

$$K(\mathbf{x}, \mathbf{t}) = H(\mathbf{x} - \mathbf{t}), \quad \mathbf{x}, \mathbf{t} \in \mathbb{R}^d$$

where $H : \mathbb{R}^d \rightarrow \mathbb{R}$ is a differentiable function

- *Radial* if it has the form

$$K(\mathbf{x}, \mathbf{t}) = h(\|\mathbf{x} - \mathbf{t}\|), \quad \mathbf{x}, \mathbf{t} \in \mathbb{R}^d$$

where $h : [0, \infty) \rightarrow [0, \infty)$ is a differentiable function

The Gaussian kernel

An important example of a radial kernel is the Gaussian kernel

$$K(\mathbf{x}, \mathbf{t}) = \exp(-\beta \|\mathbf{x} - \mathbf{t}\|^2), \quad \beta > 0, \mathbf{x}, \mathbf{t} \in \mathbb{R}^d$$

It is a kernel because it is the product of two kernels

$$K(\mathbf{x}, \mathbf{t}) = (\exp(-\beta(\mathbf{x}^\top \mathbf{x} + \mathbf{t}^\top \mathbf{t}))) \exp(2\beta \mathbf{x}^\top \mathbf{t})$$

(We saw before that $\exp(2\beta \mathbf{x}^\top \mathbf{t})$ is a kernel. Clearly $\exp(-\beta(\mathbf{x}^\top \mathbf{x} + \mathbf{t}^\top \mathbf{t}))$ is a kernel with one-dimensional feature map $\phi(\mathbf{x}) = \exp(-\beta \mathbf{x}^\top \mathbf{x})$)

Exercise:

Can you find a feature map representation for the Gaussian kernel?

Other kernels

In our examples we have mainly focused on kernels defined on \mathbb{R}^n , more generally and usefully they can be defined on other input spaces X for example.

1. Kernels between sets
 2. Kernels on text and strings
 3. Kernels between graphs
 4. Kernel between vertices on a graph
- Defining useful kernels between on new domains X allows for a host of ML algorithms to be transferred to that domain. For example ridge regression, k -NN, SVMs, k -means, PCA, etc.

Further examples (Kernels)

From *Kernel Methods for Pattern Analysis*, Shawe-Taylor.J, and Cristianini N., Cambridge University Press (2004)

Kernels (to give you an idea)

- Definition 9.1 Polynomial kernel 286
- Computation 9.6 All-subsets kernel 289
- Computation 9.8 Gaussian kernel 290
- Computation 9.12 ANOVA kernel 293
- Computation 9.18 Alternative recursion for ANOVA kernel 296
- Computation 9.24 General graph kernels 301
- Definition 9.33 Exponential diffusion kernel 307
- Definition 9.34 von Neumann diffusion kernel 307
- Computation 9.35 Evaluating diffusion kernels 308
- Computation 9.46 Evaluating randomised kernels 315
- Definition 9.37 Intersection kernel 309
- Definition 9.38 Union-complement kernel 310
- Remark 9.40 Agreement kernel 310
- Section 9.6 Kernels on real numbers 311
- Remark 9.42 Spline kernels 313
- Definition 9.43 Derived subsets kernel 313
- Definition 10.5 Vector space kernel 325
- Computation 10.8 Latent semantic kernels 332
- Definition 11.7 The p-spectrum kernel 342
- Computation 11.10 The p-spectrum recursion 343
- Remark 11.13 Blended spectrum kernel 344
- Computation 11.17 All-subsequences kernel 347
- Computation 11.24 Fixed length subsequences kernel 352
- Computation 11.33 Naive recursion for gap-weighted subsequences kernel 358
- Computation 11.36 Gap-weighted subsequences kernel 360
- Computation 11.45 Trie-based string kernels 367
- Algorithm 9.14 ANOVA kernel 294
- Algorithm 9.25 Simple graph kernels 302
- Algorithm 11.20 All-non-contiguous subsequences kernel 350
- Algorithm 11.25 Fixed length subsequences kernel 352
- Algorithm 11.38 Gap-weighted subsequences kernel 361
- Algorithm 11.40 Character weighting string kernel 364
- Algorithm 11.41 Soft matching string kernel 365
- Algorithm 11.42 Gap number weighting string kernel 366
- Algorithm 11.46 Trie-based p-spectrum kernel 368
- Algorithm 11.51 Trie-based mismatch kernel 371
- Algorithm 11.54 Trie-based restricted gap-weighted kernel 374
- Algorithm 11.62 Co-rooted subtree kernel 380
- Algorithm 11.65 All-subtree kernel 383
- Algorithm 12.8 Fixed length HMM kernel 401
- Algorithm 12.14 Pair HMM kernel 407
- Algorithm 12.17 Hidden tree model kernel 411
- Algorithm 12.34 Fixed length Markov model Fisher kernel 427

Further examples (Algorithms)

From *Kernel Methods for Pattern Analysis*, Shawe-Taylor.J, and Cristianini N., Cambridge University Press (2004)

Algorithms (to give you an idea)

Computation 2.5 Ridge regression 30
Computation 5.14 Regularised Fisher discriminant 131
Computation 5.15 Regularised kernel Fisher discriminant 133
Computation 6.3 Maximising variance 141
Computation 6.18 Maximising covariance 154
Computation 6.30 Canonical correlation analysis 163
Computation 6.32 Kernel CCA 165
Computation 6.34 Regularised CCA 169
Computation 6.35 Kernel regularised CCA 169
Computation 7.1 Smallest enclosing hypersphere 193
Computation 7.7 Soft minimal hypersphere 199
Computation 7.10 nu-soft minimal hypersphere 202
Computation 7.19 Hard margin SVM 209
Computation 7.28 1-norm soft margin SVM 216
Computation 7.36 2-norm soft margin SVM 223
Computation 7.40 Ridge regression optimisation 229
Computation 7.43 Quadratic e-insensitive SVR 231
Computation 7.46 Linear e-insensitive SVR 233
Computation 7.50 nu-SVR 235
Computation 8.8 Soft ranking 254
Computation 8.17 Cluster quality 261
Computation 8.19 Cluster optimisation strategy 265
Computation 8.25 Multiclass clustering 272
Computation 8.27 Relaxed multiclass clustering 273
Computation 8.30 Visualisation quality 277

Algorithm 5.1 Normalisation 110
Algorithm 5.3 Centering data 113
Algorithm 5.4 Simple novelty detection 116
Algorithm 5.6 Parzen based classifier 118
Algorithm 5.12 Cholesky decomposition or dual Gram-Schmidt 126
Algorithm 5.13 Standardising data 128
Algorithm 5.16 Kernel Fisher discriminant 134
Algorithm 6.6 Primal PCA 143
Algorithm 6.13 Kernel PCA 148
Algorithm 6.16 Whitening 152
Algorithm 6.31 Primal CCA 164
Algorithm 6.36 Kernel CCA 171
Algorithm 6.39 Principal components regression 175
Algorithm 6.42 PLS feature extraction 179
Algorithm 6.45 Primal PLS 182
Algorithm 6.48 Kernel PLS 187
Algorithm 7.2 Smallest hypersphere enclosing data 194
Algorithm 7.8 Soft hypersphere minimisation 201
Algorithm 7.11 nu-soft minimal hypersphere 204
Algorithm 7.21 Hard margin SVM 211
Algorithm 7.26 Alternative hard margin SVM 214
Algorithm 7.29 1-norm soft margin SVM 218
Algorithm 7.32 nu-SVM 221
Algorithm 7.37 2-norm soft margin SVM 225
Algorithm 7.41 Kernel ridge regression 229
Algorithm 7.45 2-norm SVR 232
Algorithm 7.47 1-norm SVR 234
Algorithm 7.51 nu-support vector regression 236
Algorithm 7.52 Kernel perceptron 237
Algorithm 7.59 Kernel adatron 242
Algorithm 7.61 On-line SVR 244
Algorithm 8.9 nu-ranking 254
Algorithm 8.14 On-line ranking 257
Algorithm 8.22 Kernel k-means 269
Algorithm 8.29 MDS for kernel-embedded data 276
Algorithm 8.33 Data visualisation 280

Computational Summary for ridge regression

Summary : Computation with Basis Functions

Data: X , $(m \times n)$; \mathbf{y} , $(m \times 1)$

Basis Functions: ϕ_1, \dots, ϕ_N where $\phi_i : \mathbb{R}^n \rightarrow \mathbb{R}$

Feature Map: $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^N$

$$\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_N(\mathbf{x})), \quad \mathbf{x} \in \mathbb{R}^n$$

Mapped Data Matrix:

$$\Phi := \begin{pmatrix} \phi(\mathbf{x}_1) \\ \vdots \\ \phi(\mathbf{x}_m) \end{pmatrix} = \begin{pmatrix} \phi_1(\mathbf{x}_1) & \dots & \phi_N(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ \phi_1(\mathbf{x}_m) & \dots & \phi_N(\mathbf{x}_m) \end{pmatrix}, \quad (m \times N)$$

Regression Coefficients: $\mathbf{w} = (\Phi^\top \Phi + \lambda I_N)^{-1} \Phi^\top \mathbf{y}$

Regression Function: $\hat{y}(\mathbf{x}) = \sum_{i=1}^N w_i \phi_i(\mathbf{x})$

Summary : Computation with Kernels

Data: X , $(m \times n)$; \mathbf{y} , $(m \times 1)$

Kernel Function: $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$

Kernel Matrix:

$$\mathbf{K} := \begin{pmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & \dots & K(\mathbf{x}_1, \mathbf{x}_m) \\ \vdots & \ddots & \vdots \\ K(\mathbf{x}_m, \mathbf{x}_1) & \dots & K(\mathbf{x}_m, \mathbf{x}_m) \end{pmatrix}, \quad (m \times m)$$

Regression Coefficients: $\alpha = (\mathbf{K} + \lambda I_m)^{-1} \mathbf{y}$

Regression Function: $\hat{y}(\mathbf{x}) = \sum_{i=1}^m \alpha_i K(\mathbf{x}_i, \mathbf{x})$

Suggested Readings

Chapters 2,3 (Additionally read chapter 9 for more depth). *Kernel Methods for Pattern Analysis*, Shawe-Taylor.J, and Cristianini N., Cambridge University Press (2004)

Going Deeper ...

- *Regularization Matters: Generalization and Optimization of Neural Nets v.s. their Induced Kernel.* One of series of paper on the Neural Tangent Kernel (connecting overparameterised neural networks to a particular RKHS). One of aim of this research is to get a better understanding for why NNs generalise.
- *Convolution Kernels on Discrete Structures.* Classic paper with a variety of nice ideas on Kernels for discrete structures.

Problems – 1

1. Prove results on page 9.
2. Consider the solution to linear regression optimisation problem. When is it advantageous to compute it via the primal solution? When is it advantageous to compute it via the dual solution? Explain why
3. Given a kernel $K : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R}$ where $K(\mathbf{x}, \mathbf{t}) := (1 + \langle \mathbf{x}, \mathbf{t} \rangle)^2$. Find a feature map $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^6$ which corresponds to the kernel.
4. For each of the following functions $K : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R}$ argue whether they are a valid kernel (i.e. the kernel can be written as an inner product in some feature space) and when the answer is positive derive an associated feature map representation.

4.1 $K(\mathbf{x}, \mathbf{t}) = \mathbf{x}^\top D\mathbf{t}$, where D is the matrix

$$\begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix}$$

4.2 $K(\mathbf{x}, \mathbf{t}) = \mathbf{x}^\top D\mathbf{t}$, where D is the matrix

$$\begin{bmatrix} -1 & 2 \\ 2 & 4 \end{bmatrix}$$

4.3 $K(\mathbf{x}, \mathbf{t}) = \exp(x_1 t_1)$, where x_1 is the first component of the vector \mathbf{x} and, likewise, t_1 is the first component of the vector \mathbf{t} .

4.4 $K(\mathbf{x}, \mathbf{t}) = \mathbf{x}^\top \mathbf{t} - (\mathbf{x}^\top \mathbf{t})^2$.

4.5 Now prove that if $K : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R}$ is a given valid kernel then the following transformed kernels are also valid:

4.5.1 $K(A\mathbf{x}, A\mathbf{t})$, where A is a given 2×2 matrix.

4.5.2 $f(\mathbf{x})K(\mathbf{x}, \mathbf{t})f(\mathbf{t})$, where f is a given real-valued function.

5. Consider a Gaussian kernel function $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ defined by $K(\mathbf{x}, \mathbf{z}) := e^{-\|\mathbf{x}-\mathbf{z}\|^2}$, does there exist a finite-dimensional feature map representation? I.e., does there exist a $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^d$ such that $K(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$? Indicate an answer "yes" or "no" and provide an argument supporting your answer. [hard]

Problems – 2

1. Argue that the vector space definition implies that every element $\mathbf{x} \in X$ has an additive inverse $-\mathbf{x} \in X$ such that $\mathbf{x} + (-\mathbf{x}) = 0$
2. *Kernels between sets.* Let X be a finite set define $K : 2^X \times 2^X \rightarrow \mathbb{R}$ as

$$K(A, B) := 2^{|A \cap B|}$$

where $A, B \subseteq X$. Prove that K is a kernel.

3. *Min Kernel.*

3.1 Argue that $\min(x, t)$ (where $x, t \in [0, \infty)$) is a kernel for “a more complicated example” on page 9. See discussion on 41, on going from a kernel to a Hilbert space. [technical]

3.2 *Determining an explicit feature map.* Find a set of basis functions $\phi_i : [0, \infty) \rightarrow \mathbb{R}$ ($i = 1, 2, \dots, \infty$) such that

$$\min(x, t) = \sum_{i=1}^{\infty} \phi_i(x)\phi_i(t)$$

[technical, **very difficult**]

3. Support Vector Machines

COMP0078: Supervised Learning

Mark Herbster

18 October 2021

University College London
Department of Computer Science
svm21v3

Acknowledgments and References

Thanks

Thanks to Massi Pontil for many of the slides.

Today

- Optimal Separating Hyperplane
- Soft Margin Separation
- Support Vector Machines
- Connection to Regularisation

Optimal Separating Hyperplane

Part I Optimal Separating Hyperplane

Separating hyperplane – 1

Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m \in \mathbf{R}^n \times \{-1, 1\}$ be a training set

By a **hyperplane** we mean a set $H_{\mathbf{w}, b} = \{\mathbf{x} \in \mathbf{R}^n : \mathbf{w}^\top \mathbf{x} + b = 0\}$ (affine linear space) parameterized by $\mathbf{w} \in \mathbf{R}^n$ and $b \in \mathbf{R}$

We assume that the data are linearly separable, that is, there exist $\mathbf{w} \in \mathbf{R}^n$ and $b \in \mathbf{R}$ such that

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) > 0, \quad i = 1, \dots, m \tag{1}$$

in which case we call $H_{\mathbf{w}, b}$ a **separating hyperplane**

Note that we require the inequality in eq.(1) to be strict (we do not admit that the data lie on a hyperplane)

Separating hyperplane – 2

The distance $\rho_x(\mathbf{w}, b)$ of a point \mathbf{x} from a hyperplane $H_{\mathbf{w}, b}$ is

$$\rho_x(\mathbf{w}, b) := \frac{|\mathbf{w}^\top \mathbf{x} + b|}{\|\mathbf{w}\|}$$

If $H_{\mathbf{w}, b}$ separates the training set S we define its **margin** as

$$\rho_S(\mathbf{w}, b) := \min_{i=1}^m \rho_{x_i}(\mathbf{w}, b)$$

If $H_{\mathbf{w}, b}$ is a hyperplane (separating or not) we also define the *margin of a point \mathbf{x}* as $\frac{\mathbf{w}^\top \mathbf{x} + b}{\|\mathbf{w}\|}$ (note that this can be positive or negative)

Separating hyperplane – 3

Let's verify the observations on the previous slide.

The projection from a point \mathbf{x} to $H_{\mathbf{w}, b}$ is

$$\mathbf{p} = \mathbf{x} - \frac{\mathbf{w}(b + \langle \mathbf{w}, \mathbf{x} \rangle)}{\|\mathbf{w}\|^2}$$

To verify this we check that i) \mathbf{p} is on the hyperplane and ii) $\mathbf{x} - \mathbf{p}$ is orthogonal to $\mathbf{p} - \mathbf{x}'$ where \mathbf{x}' is some other point on the hyperplane $H_{\mathbf{w}, b}$.

Verifying i: We observe that,

$$\langle \mathbf{w}, \mathbf{p} \rangle + b = \langle \mathbf{w}, \mathbf{x} \rangle - \frac{\langle \mathbf{w}, \mathbf{w} \rangle(b + \langle \mathbf{w}, \mathbf{x} \rangle)}{\|\mathbf{w}\|^2} + b = 0.$$

Verifying ii: We observe that

$$\begin{aligned} \langle \mathbf{p} - \mathbf{x}, \mathbf{p} - \mathbf{x}' \rangle &= \left\langle -\frac{\mathbf{w}(b + \langle \mathbf{w}, \mathbf{x} \rangle)}{\|\mathbf{w}\|^2}, \mathbf{p} - \mathbf{x}' \right\rangle \\ &= \frac{(b + \langle \mathbf{w}, \mathbf{x} \rangle)^2}{\|\mathbf{w}\|^2} - \left\langle \mathbf{x} - \mathbf{x}', \frac{\mathbf{w}(b + \langle \mathbf{w}, \mathbf{x} \rangle)}{\|\mathbf{w}\|^2} \right\rangle \\ &= 0 \quad \% \text{ since } \langle \mathbf{w}, \mathbf{x}' \rangle + b = 0. \end{aligned}$$

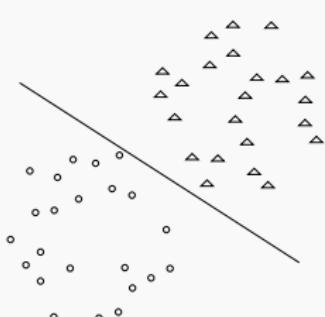
Computing the distance $\rho_x(\mathbf{w}, b) = \|\mathbf{x} - \mathbf{p}\|$ of a point \mathbf{x} from a hyperplane $H_{\mathbf{w}, b}$ is

$$\sqrt{\langle \mathbf{p} - \mathbf{x}, \mathbf{p} - \mathbf{x} \rangle} = \sqrt{\left\langle \frac{\mathbf{w}(b + \langle \mathbf{w}, \mathbf{x} \rangle)}{\|\mathbf{w}\|^2}, \frac{\mathbf{w}(b + \langle \mathbf{w}, \mathbf{x} \rangle)}{\|\mathbf{w}\|^2} \right\rangle} = \frac{|b + \langle \mathbf{w}, \mathbf{x} \rangle|}{\|\mathbf{w}\|}.$$

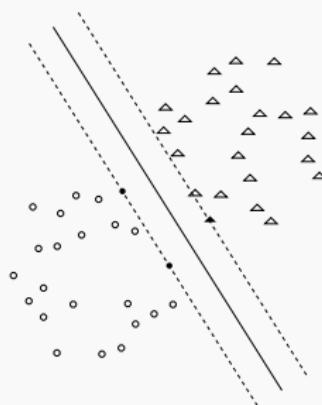
Optimal separating hyperplane (OSH)

This is the separating hyperplane with maximum margin. It solves the optimization problem

$$\rho(S) := \max_{\mathbf{w}, b} \min_{1 \leq i \leq m} \left\{ \frac{y_i(\mathbf{w}^\top \mathbf{x}_i + b)}{\|\mathbf{w}\|} : y_j(\mathbf{w}^\top \mathbf{x}_j + b) \geq 0, j = 1, \dots, m \right\} > 0$$



(a)



(b)

Choosing a parameterization

A separating hyperplane is parameterized by (\mathbf{w}, b) , but this choice is not unique (rescaling with a positive constant gives the same separating hyperplane). Two possible ways to fix the parameterization:

- *Normalized hyperplane*: set $\|\mathbf{w}\| = 1$, in which case
 $\rho_x(\mathbf{w}, b) = |\mathbf{w}^\top \mathbf{x} + b|$ and $\rho_S(\mathbf{w}, b) = \min_{i=1}^m y_i(\mathbf{w}^\top \mathbf{x}_i + b)$
- *Canonical hyperplane*: choose $\|\mathbf{w}\|$ such that $\rho_S(\mathbf{w}, b) = \frac{1}{\|\mathbf{w}\|}$, i.e.
we require that $\min_{i=1}^m y_i(\mathbf{w}^\top \mathbf{x}_i + b) = 1$ (a data-dependent parameterization)

We will mainly work with the second parameterization

Optimal separating hyperplane

- If we work with normalized hyperplanes we have

$$\rho(S) = \max_{\mathbf{w}, b} \min_i \{y_i(\mathbf{w}^\top \mathbf{x}_i + b) : y_j(\mathbf{w}^\top \mathbf{x}_j + b) \geq 0, \|\mathbf{w}\| = 1, j \in [m]\}$$

- If we work with canonical hyperplanes, instead, we have

$$\begin{aligned}\rho(S) &= \max_{\mathbf{w}, b} \left\{ \frac{1}{\|\mathbf{w}\|} : \min_i \{y_i(\mathbf{w}^\top \mathbf{x}_i + b)\} = 1, y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 0 \right\} \\ &= \max_{\mathbf{w}, b} \left\{ \frac{1}{\|\mathbf{w}\|} : y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \right\} \\ &= \frac{1}{\min_{\mathbf{w}, b} \{\|\mathbf{w}\| : y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1\}}\end{aligned}$$

Optimal separating hyperplane (cont.)

We choose to work with canonical hyperplanes and, so, look at the optimization problem

Problem P1

$$\text{Minimize} \quad \frac{1}{2} \mathbf{w}^\top \mathbf{w} \quad (\mathbf{w} \in \mathbb{R}^n)$$

$$\text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad i = 1, \dots, m,$$

The quantity $1/\|\mathbf{w}\|$ is the **margin** of the OSH

Solving a constrained optimisation with a Lagrangian

Problem

Minimise a convex function subject to linear inequality constraints

$$f(\mathbf{z}) : A\mathbf{z} \leq \mathbf{c}$$

Where $f : \mathbf{R}^n \rightarrow \mathbf{R}$, is differentiable and the matrix A is $m \times n$ and $\mathbf{c} \in \mathbf{R}^m$.

If the optimisation is **feasible** that is $\{\mathbf{z} : A\mathbf{z} \leq \mathbf{c}\}$ is non-empty then we may solve by forming the Lagrangian,

$$L(\mathbf{z}, \boldsymbol{\alpha}) := f(\mathbf{z}) + \boldsymbol{\alpha}^\top (A\mathbf{z} - \mathbf{c}),$$

and we have that

$$\max_{\boldsymbol{\alpha} \geq 0} \min_{\mathbf{z}} L(\mathbf{z}, \boldsymbol{\alpha}) = \min_{\mathbf{z}} f(\mathbf{z}) : A\mathbf{z} \leq \mathbf{c}.$$

Necessary and sufficient conditions (KKT) for a solution $(\bar{\boldsymbol{\alpha}}, \bar{\mathbf{z}})$

1. $A\bar{\mathbf{z}} \leq \mathbf{c}$
2. $\bar{\boldsymbol{\alpha}} \geq 0$
3. $\nabla_{\mathbf{z}} L|_{\bar{\mathbf{z}}} = 0$
4. $(A\bar{\mathbf{z}} - \mathbf{c})_i \bar{\alpha}_i = 0 \quad i = 1, \dots, m$ ("complementary slackness")

Saddle point

The solution of problem **P1** is equivalent to determine the **saddle point** of the Lagrangian function

$$L(\mathbf{w}, b; \alpha) = \frac{1}{2} \mathbf{w}^\top \mathbf{w} - \sum_{i=1}^m \alpha_i [y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1] \quad (2)$$

where $\alpha_i \geq 0$ are the Lagrange multipliers

We minimize L over (\mathbf{w}, b) and maximise over α with $\alpha \geq \mathbf{0}$.

Differentiating w.r.t \mathbf{w} and b we obtain:

$$\frac{\partial L}{\partial b} = - \sum_{i=1}^m y_i \alpha_i = 0 \quad (3)$$

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i = \mathbf{0} \Rightarrow \mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i \quad (4)$$

Dual problem

Expanding (2) we have

$$\frac{1}{2} \underbrace{\alpha^\top A \alpha}_{\mathbf{w}^\top \mathbf{w}} - \underbrace{\sum_{i=1}^m \alpha_i y_i \mathbf{w}^\top \mathbf{x}_i}_{\mathbf{A}\alpha} - b \underbrace{\sum_{i=1}^m \alpha_i y_i}_{0} + \underbrace{\sum_{i=1}^m \alpha_i}_{0}$$

Substituting (3) and (4) in eq.(2) and defining the $m \times m$ matrix $\mathbf{A} := (y_i y_j \mathbf{x}_i^\top \mathbf{x}_j : i, j = 1, \dots, m)$ leads to the **dual problem**.

Problem P2

$$\text{Maximize } Q(\alpha) := -\frac{1}{2} \alpha^\top \mathbf{A} \alpha + \sum_i \alpha_i$$

$$\begin{aligned} \text{subject to } & \sum_i y_i \alpha_i = 0 \\ & \alpha_i \geq 0, \quad i = 1, \dots, m \end{aligned}$$

Note that the complexity of this problem depends on m , not on the number of input components n (same as ridge regression)

Kuhn-Tucker conditions and support vectors

If $\bar{\alpha}$ is a solution of the dual problem then the solution $(\bar{\mathbf{w}}, \bar{b})$ of the primal problem is given by

$$\bar{\mathbf{w}} = \sum_{i=1}^m \bar{\alpha}_i y_i \mathbf{x}_i$$

Note that $\bar{\mathbf{w}}$ is a linear combination of only the \mathbf{x}_i for which $\bar{\alpha}_i > 0$. These \mathbf{x}_i are termed **support vectors** (SVs)

Parameter \bar{b} can be determined by looking at the Kuhn-Tucker conditions (“complementary slackness”)

$$\bar{\alpha}_i (y_i (\bar{\mathbf{w}}^\top \mathbf{x}_i + \bar{b}) - 1) = 0$$

Specifically if \mathbf{x}_j is a SV we have that

$$\bar{b} = y_j - \bar{\mathbf{w}}^\top \mathbf{x}_j$$

Some remarks

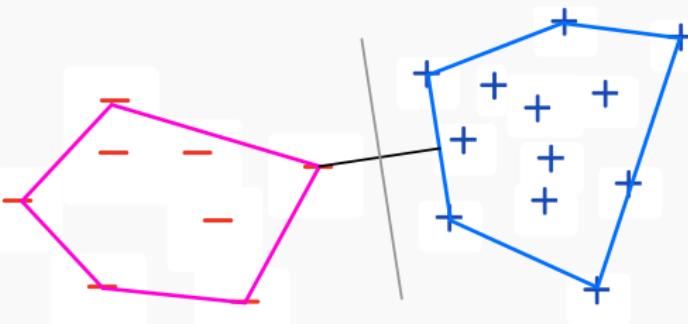
- The fact that the OSH is determined only by the SVs is most remarkable. Usually, the support vectors are a small subset of the training data
- All the information contained in the data set is summarized by the support vectors: The whole data set could be replaced by only these points and the **same** hyperplane would be found
- A new point \mathbf{x} is classified as $\text{sgn} \left(\sum_{i=1}^m y_i \bar{\alpha}_i \mathbf{x}_i^\top \mathbf{x} + \bar{b} \right)$

Connection between number of support vectors and generalization

Let n_{SV} equal the expected number of support vectors in an SVM trained on m examples sampled IID then the expected generalization error of an SVM trained on $m - 1$ examples is bounded above by n_{SV}/m .

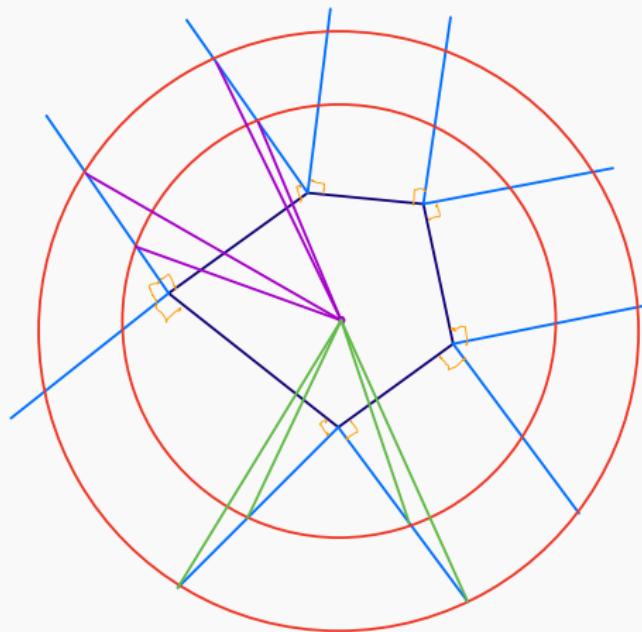
Intuitions – 1

1. We've argued qualitatively that **sparsity** of support vectors implies good generalisation performance.
2. We've argued intuitively that large margin implies good generalisation performance. Note: *there are a number of arguments in literature which prove qualitative bounds on the generalisation error based on the margin but for our purposes these are beyond the scope of the class.*
3. We will give an intuitive argument that all else being equal a larger margin implies fewer support vectors.
4. The SVM optimisation problem can be shown to be equivalent to finding the (a) shortest line segment that connects the hull of the convex positive points to the negative points and then the perpendicular bisecting hyperplane of the line is the linear classifier.



Intuitions – 2

1. The number of support vectors associated with a **face** is the number of vertices.
2. All else being equal a point nearer to a convex polytope tend to be “nearer” to a larger dimensional face than a “farther” point.



Linearly nonseparable case – 1

Ideally we would like to minimise

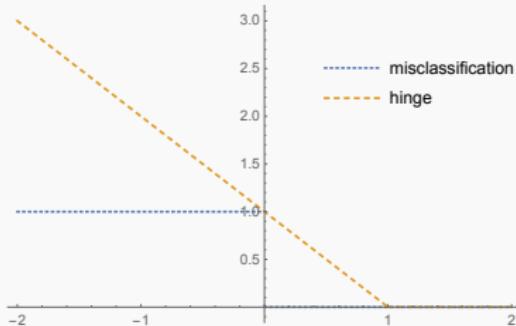
$$\frac{1}{2} \mathbf{w}^\top \mathbf{w} + C \sum_{i=1}^m V_{\text{mc}}(y_i, \mathbf{w}^\top \mathbf{x}_i + b)$$

However this is known to be **NP-hard!** So instead we convexify by replacing the misclassification loss by the hinge loss

$$\frac{1}{2} \mathbf{w}^\top \mathbf{w} + C \sum_{i=1}^m V_{\text{hinge}}(y_i, \mathbf{w}^\top \mathbf{x}_i + b) \quad (5)$$

$$V_{\text{mc}}(y, \hat{y}) = \mathcal{I}[y = \text{sign}(\hat{y})]$$

$$V_{\text{hinge}}(y, \hat{y}) = \max(0, 1 - y\hat{y})$$



$$V_{\text{mc}}(\mathbf{1}, \hat{y}), \text{ and } V_{\text{hinge}}(\mathbf{1}, \hat{y}), .$$

We now have a convex (quadratic) optimisation problem.

Linearly nonseparable case

Observe we can rewrite (5) as

Problem P3

$$\text{Minimize} \quad \frac{1}{2} \mathbf{w}^\top \mathbf{w} + C \sum_{i=1}^m \xi_i$$

$$\begin{aligned} \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \dots, m \end{aligned}$$

Note: The slack variables ξ_i relax the separation constraints ($\xi_i > 0 \Rightarrow y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 1$ i.e., the margin is less than 1).

Saddle point

The solution of problem **P3** is equivalent to determining the **saddle point** of the Lagrangian function

$$L(\mathbf{w}, \xi, b; \alpha, \beta) = \frac{1}{2} \mathbf{w}^\top \mathbf{w} + C \sum_{i=1}^m \xi_i - \sum_{i=1}^m \alpha_i [y_i (\mathbf{w}^\top \mathbf{x}_i + b) + \xi_i - 1] - \sum_{i=1}^m \beta_i \xi_i \quad (6)$$

where $\alpha_i, \beta_i \geq 0$ are the Lagrange multipliers

We minimize L over (\mathbf{w}, ξ, b) and maximize over α, β with $\alpha, \beta \geq 0$.

Differentiating w.r.t \mathbf{w} , ξ , and b we obtain:

$$\begin{aligned} \frac{\partial L}{\partial b} &= - \sum_{i=1}^m y_i \alpha_i = 0 \\ \frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i = \mathbf{0} \Rightarrow \mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i \\ \frac{\partial L}{\partial \xi_i} &= C - \alpha_i - \beta_i = 0 \Rightarrow 0 \leq \alpha_i \leq C \end{aligned} \quad (7)$$

New dual problem

Analogous to **(P2)** substituting (7) into (6) leads to the dual problem.

Problem **P4**

$$\text{Maximize} \quad Q(\alpha) := -\frac{1}{2}\alpha^\top A\alpha + \sum_i \alpha_i$$

$$\begin{aligned} \text{subject to} \quad & \sum_i y_i \alpha_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, m \end{aligned}$$

This is like problem **P2** except that now we have “box constraints” on α_i .
If the data is linearly separable, by choosing C large enough we obtain
the OSH

Nonseparable case (cont)

Again we have

$$\bar{\mathbf{w}} = \sum_{i=1}^m \bar{\alpha}_i y_i \mathbf{x}_i,$$

while \bar{b} can be determined from $\bar{\alpha}$, solution of the problem **P4**, and from the new Kuhn-Tucker conditions (“complementary slackness”)

$$\bar{\alpha}_i (y_i (\bar{\mathbf{w}}^\top \mathbf{x}_i + \bar{b}) - 1 + \bar{\xi}_i) = 0 \quad (*)$$

$$(C - \bar{\alpha}_i) \bar{\xi}_i = 0 \quad (**)$$

Where $(**)$ follows since $\beta_i = C - \alpha_i$. Again, points for which $\bar{\alpha}_i > 0$ are termed **support vectors**

A closer look at the KKT conditions

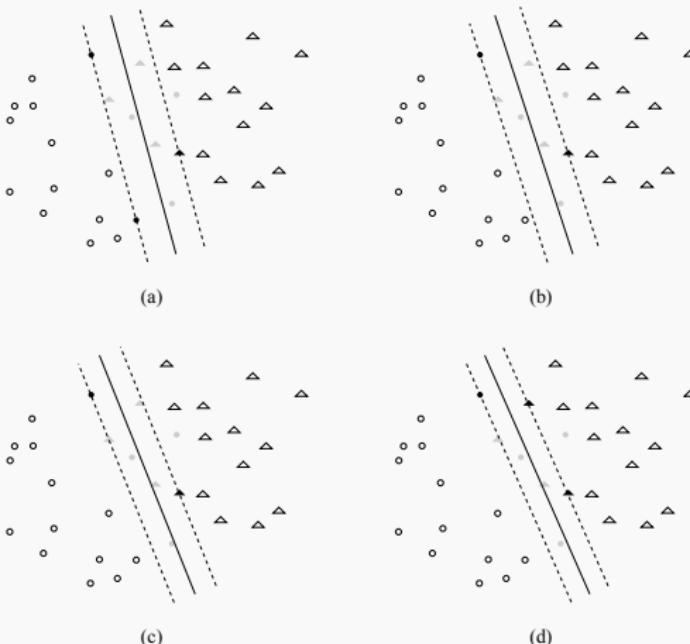
Equation (*) and (**) tell us that if

- $y_i(\bar{\mathbf{w}}^\top \mathbf{x}_i + \bar{b}) > 1 \Rightarrow \bar{\alpha}_i = 0$ (not a SV)
- $y_i(\bar{\mathbf{w}}^\top \mathbf{x}_i + \bar{b}) < 1 \Rightarrow \bar{\alpha}_i = C$ (a SV with positive slack $\bar{\xi}_i$)
- $y_i(\bar{\mathbf{w}}^\top \mathbf{x}_i + \bar{b}) = 1 \Rightarrow \bar{\alpha}_i \in [0, C]$ (if $\bar{\alpha}_i > 0$ a SV “on the margin”)

Remark: Conversely, from eqs.(*),(**) if $\bar{\alpha}_i = 0$ then

$y_i(\bar{\mathbf{w}}^\top \mathbf{x}_i + \bar{b}) \geq 1, \bar{\xi}_i = 0$; if $\bar{\alpha}_i \in (0, C)$ then $y_i(\bar{\mathbf{w}}^\top \mathbf{x}_i + \bar{b}) = 1, \bar{\xi}_i = 0$; if $\bar{\alpha}_i = C$ then $y_i(\bar{\mathbf{w}}^\top \mathbf{x}_i + \bar{b}) \leq 1, \bar{\xi}_i \geq 0$

The role of the parameter C



Optimal separating hyperplane for four increasing values of C . Both the margin and the training error are non-increasing functions of C

The role of the parameter C (cont.)

The parameter C controls the trade-off between $\|\mathbf{w}\|^2$ and the training error $\sum_{i=1}^m \xi_i$

It can be shown that the optimal value of the Lagrange multipliers $\bar{\alpha}_i$ (and, so, $\bar{\mathbf{w}}, \bar{b}$) are piecewise quadratic functions of C . This helps re-computing the solution when varying C .

- C is often selected by minimizing the leave-one-out (LOO) cross validation error.

Observations on computing LOO

1. We need “retrain” the SVM no more than **#SVs times (Why?)** – retraining is “fast.”
2. Alternatively observe that rather than compute LOO one could use $\frac{\#SVs}{m}$ as an upper bound on the LOO CV error.

Support Vector Machines (SVMs)

The above analysis holds true if we work with a feature map $\phi : \mathcal{X} \rightarrow \mathcal{W}$. We simply replace \mathbf{x} by $\phi(\mathbf{x})$ and $\mathbf{x}^\top \mathbf{t}$ by $\langle \phi(\mathbf{x}), \phi(\mathbf{t}) \rangle = K(\mathbf{x}, \mathbf{t})$

An SVM with kernel K is the function

$$f(\mathbf{x}) = \sum_{i=1}^m y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b, \quad \mathbf{x} \in \mathcal{X}$$

where the parameters α_i solve problem **P4** with

$\mathbf{A} = (y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) : i, j = 1, \dots, m)$ and b is obtained as discussed above

A new point $\mathbf{x} \in \mathcal{X}$ is classified as $\text{sgn}(f(\mathbf{x}))$

Connection to regularization

The SVM formulation above is equivalent to the problem

$$E_\lambda(\mathbf{w}, b) = \sum_{i=1}^m \max(1 - y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle + b), 0) + \lambda \|\mathbf{w}\|^2$$

with $\lambda = \frac{1}{2C}$

In fact, we have

$$\begin{aligned} & \min_{\mathbf{w}, b, \xi} \left\{ C \sum_{i=1}^m \xi_i + \frac{1}{2} \|\mathbf{w}\|^2 : y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i, \xi_i \geq 0 \right\} = \\ & \min_{\mathbf{w}, b} \left\{ \min_{\xi} \left\{ C \sum_{i=1}^m \xi_i + \frac{1}{2} \|\mathbf{w}\|^2 : \xi_i \geq 1 - y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle + b), \xi_i \geq 0 \right\} \right\} = \\ & \min_{\mathbf{w}, b} \left\{ C \sum_{i=1}^m \max(1 - y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle + b), 0) + \frac{1}{2} \|\mathbf{w}\|^2 \right\} = CE_{\frac{1}{2C}}(\mathbf{w}, b) \end{aligned}$$

SVM regression

SVM's can be developed for regression as well. Here we choose the loss
 $= |y - f(\mathbf{x})|_\epsilon = \max(|y - f(\mathbf{x})| - \epsilon, 0)$

$$\text{Minimize} \quad \frac{1}{2} \mathbf{w}^\top \mathbf{w} + C \sum_{i=1}^m (\xi_i + \xi_i^*)$$

$$\begin{aligned} \text{subject to} \quad & \mathbf{w}^\top \mathbf{x}_i + b - y_i \leq \epsilon + \xi_i, \\ & y_i - \mathbf{w}^\top \mathbf{x}_i - b \leq \epsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0, \quad i = 1, \dots, m \end{aligned}$$

SVMs loss functions (both for classification and regression) are **scale sensitive**: errors below a certain resolution do not count. This leads to sparse solutions!

Solution methods

The above optimization problems are Quadratic Programming (QP) problems. Several methods (eg, interior point methods) from convex optimization exist for solving QP problems

If we work with a non-linear kernel, the number of underlying features, N , is typically much larger (or infinite) than the number of examples. Thus, we need to solve the dual problem

However, if $m \gg N$ it is more efficient to solve the primal problem

Decomposition of the dual problem

For large datasets (say $m > 10^5$) it is practically impossible to solve the dual problem with standard optimization techniques (matrix \mathbf{A} is dense!)

A typical approach is to iteratively optimize wrt. an “active set” \mathcal{A} of variables. Set $\alpha = 0$, choose $q \leq m$ and a subset \mathcal{A} of q variables, $\mathcal{A} = \{\alpha_{i_1}, \dots, \alpha_{i_q}\}$. We repeat until convergence:

- Optimize $Q(\alpha)$ wrt. the variables in \mathcal{A}
- Remove one variable from \mathcal{A} which satisfies the KKT conditions and add one variable, if any, which violates the KKT conditions. If no such variable exists stop

One can show that after each iteration Q increases

Suggested Readings

Recommended: *The Elements of Statistical Learning* ..., Chapters 12.1-12.3.

Background on Lagrangian and KKT conditions see Chapter 5 *Convex Optimization* by Boyd and Vandenberghe.

Going Deeper ...

- *Making Large-Scale SVM Learning Practical.* This is an early work on efficient optimization for SVMs. The state of the art has considerably advanced since this paper but it covers the elementary concepts.
- *When Do Neural Networks Outperform Kernel Methods?*. For what tasks are NNs good? For what tasks do Kernel Methods do well? This paper consider this question in terms of potential latent low-dimensional representations of the data.

Problems – 1

1. Describe the criterion used by hard margin Support Vector Machines to choose a separating hyperplane for a linearly separable dataset, illustrating with a diagram indicating the margin and the support vectors.
2. For a hard-margin SVM. If we remove one of
 - 2.1 the examples which is a support vector from the training set,
 - 2.2 the examples which is not a support vector from the training setand retrain with out that example. How does the maximum margin change?
3. Consider the optimisation given by

$$\min_{\mathbf{w}, b, \gamma, \xi} \quad \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i$$

$$\text{subject to} \quad y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1 - \xi_i, \quad \xi_i \geq 0, \\ i = 1, \dots, m.$$

Which part of the optimisation corresponds to the regulariser? What is the loss function incorporated into the optimisation?

Problems – 2

1. Assume that the set $S = \{(x_i, y_i)\}_{i=1}^m \subset \mathbf{R}^2 \times \{-1, 1\}$ of binary examples is strictly linearly separable by a line going through the origin, that is, there exists a vector $\mathbf{w} \in \mathbf{R}^2$ such that the linear function $f(x) = \mathbf{w}^\top x$, $x \in \mathbf{R}^2$ has the property that $y_i f(x_i) > 0$ for every $i = 1, \dots, m$. Consider the optimisation problem (linearly separable SVM):

$$\mathbf{P1} : \quad \text{minimise} \left\{ \frac{1}{2} \mathbf{w}^\top \mathbf{w} : y_i \mathbf{w}^\top \mathbf{x}_i \geq 1, i = 1, \dots, m \right\}.$$

Argue that the above problem has a unique solution. Describe the geometric meaning of this solution.

2. Show that the vector \mathbf{w} solving problem **P1** has the form $\mathbf{w} = \sum_{i=1}^m c_i y_i \mathbf{x}_i$ where c_1, \dots, c_m are some nonnegative coefficients. [HINT: use the method of Lagrange multipliers]
3. Show that the coefficients c_1, \dots, c_m in the above formula solve the optimization problem

$$\mathbf{P2} : \quad \max \left\{ -\frac{1}{2} \sum_{i,j=1}^m c_i c_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + \sum_{i=1}^m c_i : c_j \geq 0, j = 1, \dots, m \right\}.$$

Finally, if $(\hat{c}_1, \dots, \hat{c}_m)$ is the solution to this problem and $\hat{\mathbf{w}}$ is the solution to problem **P1**, argue that $\hat{\mathbf{w}}^\top \hat{\mathbf{w}} = \sum_{i=1}^m \hat{c}_i$.

4. Tree-based learning and ensemble methods

COMP0078: Supervised Learning

Mark Herbster

25 October 2021

University College London
Department of Computer Science
tree21v3

Acknowledgments and References

Thanks

Thanks to Massi Pontil and John Shawe-Taylor for many of the slides.

Today

- Classification and regression trees (CART)
- Ensemble Learning
- Bagging
- Adaboost

Tree-based learning algorithms

Part I

Tree-based learning algorithms

Tree-based learning algorithms

Tree methods attempt to partition the input space into a set of rectangles and fit a simple model (e.g. a constant) in each one

$$f(\mathbf{x}) = \sum_{p=1}^P c_p \mathcal{I}[\mathbf{x} \in R_p]$$

- We partition the input space with hyper-rectangles

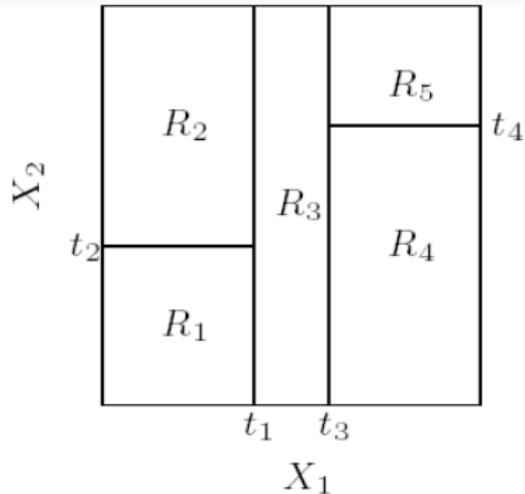
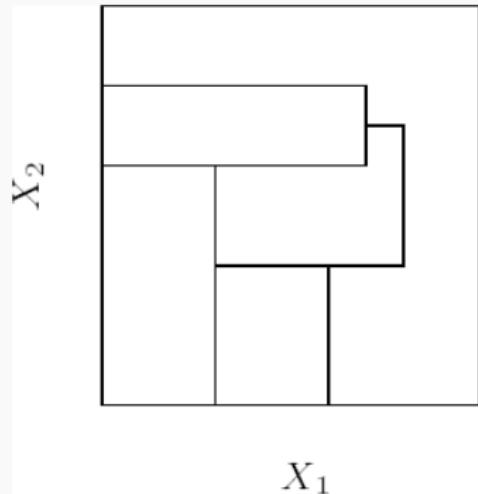
$$R_1, R_2, \dots, R_P$$

where $\bigcup_{p=1}^P R_p = \mathbf{R}^n$ and $R_a \cap R_b = \emptyset$ if $a \neq b$.

- Define $\mathcal{I}[\mathbf{x} \in R_p] = 1$ if $\mathbf{x} \in R_p$ and 0 otherwise (indicator function)
- $\{c_p\}_{p=1}^P$: some real parameters. A natural choice is

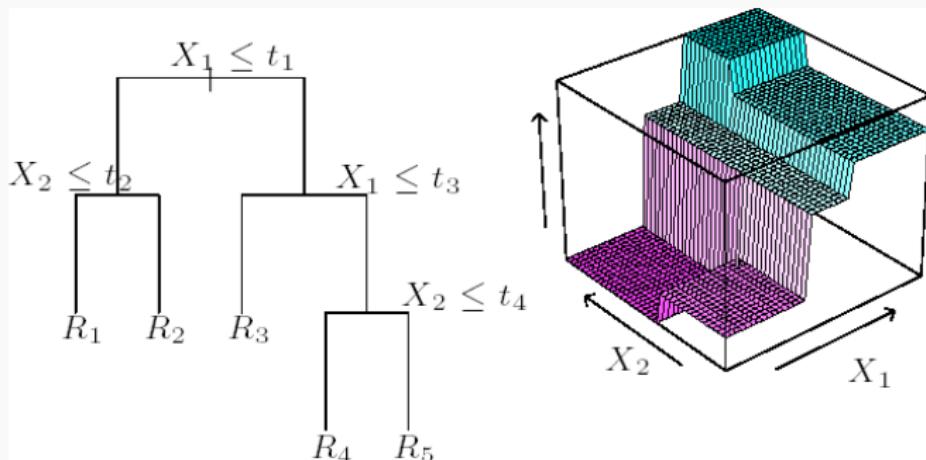
$$c_p = \text{ave}(y_i | \mathbf{x}_i \in R_p) \equiv \frac{\sum_{i=1}^m y_i \mathcal{I}[\mathbf{x}_i \in R_p]}{\sum_{i=1}^m \mathcal{I}[\mathbf{x}_i \in R_p]}$$

Recursive binary partitions



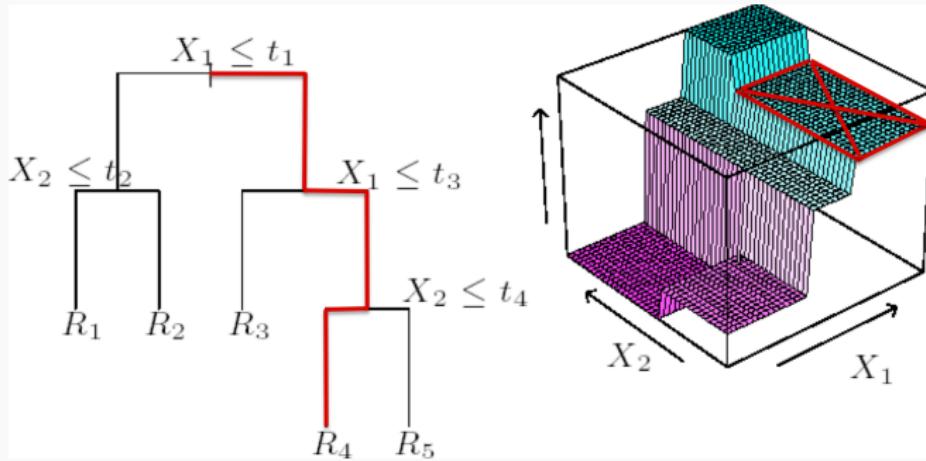
- Left plot: Each partition line has a simple description, yet partitions may be complicated to describe
- Right plot: recursive binary partition (first split the space into two regions then iterate in each region)

Tree representation – 1



- Left plot: tree representation for the recursive binary partition above
- Right plot: prediction function $f(\mathbf{x}) = \sum_{p=1}^5 c_p \mathcal{I}[\mathbf{x} \in R_p]$

Tree representation – 2



- Observe that each rectangle is defined by a conjunction.
- Thus

$$R_4 := \{\mathbf{x} : (x_1 > t_1) \wedge (x_1 > t_3) \wedge (x_2 \leq t_4)\}$$

Some remarks

- A nice feature of a recursive binary tree is its **interpretability** (e.g. in medical science the tree stratifies the population into strata of high and low outcome on the basis of patient characteristics)
- Warning: risk for **overfitting!** (with enough splits the tree can fit arbitrarily well the data)

Key question: how to compute the tree (hence the regions R_n) and how to control overfitting?

Regression trees (I)

The algorithm needs to automatically decide on the splitting variables (1st or 2nd in above example) and split points. Recall that:

$$c_p = \text{ave}(y_i | \mathbf{x}_i \in R_p)$$

which corresponds to minimizing the square error in region p

Ideally we would like to solve the problem

$$\min_{R_1, \dots, R_P} \left\{ \sum_{i=1}^m \left(y_i - \sum_{p=1}^P \text{ave}(y_j | \mathbf{x}_j \in R_p) \mathcal{I}[\mathbf{x}_i \in R_p] \right)^2 \right\}$$

But it appears to be computationally intractable. So we resort to an alternate heuristic procedure.

Regression trees (II)

Let's find the first split in the tree

Define the pair of axis parallel half-planes:

$$R_1(j, s) = \{\mathbf{x} | x_j \leq s\}, \quad R_2(j, s) = \{\mathbf{x} | x_j > s\}$$

and search for optimal values \bar{j} and \bar{s} which solve the problem

$$\min_{j,s} \left\{ \min_{c_1} \sum_{\mathbf{x}_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{\mathbf{x}_i \in R_2(j,s)} (y_i - c_2)^2 \right\}$$

Regression trees (III)

$$\min_{j,s} \left\{ \min_{c_1} \sum_{\mathbf{x}_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{\mathbf{x}_i \in R_2(j,s)} (y_i - c_2)^2 \right\}$$

- The inner minimization is solved by

$$\bar{c}_1 = \text{ave}(y_i | \mathbf{x}_i \in R_1(j,s)) \quad \bar{c}_2 = \text{ave}(y_i | \mathbf{x}_i \in R_2(j,s))$$

- For each splitting variable j the search for the best split point s can be done in $O(m)$ computations (a split may occur between any two training points)

Thus the problem is solved in $O(nm)$ computations

Regression trees (IV)

In order to build the tree we recursively assign training points to each obtained region and repeat the above steps in each one

If we do not stop the process we clearly overfit the data! So, when should we stop it?

- follow a split only if it decreases the empirical error more than a threshold? No, there might be better splits below a "bad" node
- when a maximum depth of the tree is reached? No, this could underfit or overfit. We need to look at the data to determine a good size of the tree

Regression trees (V)

We choose the tree **adaptively** from the data:

- first grow a large tree \hat{T} (stopping when a maximum number of data (e.g. 5) is assigned at each node)
- then prune the tree using a **cost-complexity pruning**, i.e. look for a subtree $T_\lambda \subseteq \hat{T}$ which minimizes

$$C_\lambda(T) = \sum_{p=1}^{|T|} m_p Q_p(T) + \lambda |T|$$

where T is a subtree of \hat{T} ,

p runs over leaf nodes of T (a subset of the nodes of \hat{T})

m_p is the number of data points assigned to node p and

$Q_p(T) = \frac{1}{m_p} \sum_{\mathbf{x}_i \in R_p} (y_i - c_p)^2$ (so the first term in C_λ is just the training error)

- Can show that there is a unique $T_\lambda \subseteq \hat{T}$ which minimizes C_λ

Regression trees (VI)

C_λ is a kind of regularized empirical error: $T_0 = \hat{T}$, the original tree.
Large values of λ result in smaller trees. A good value of λ can be obtained by cross validation

- **weakest link pruning:** successively collapse the internal nodes that produce the smallest per node increase in

$$\sum_{p=1}^{|T|} m_p Q_p(T)$$

and continue till the root (single-node) tree is produced

- search along the produced list of trees for the one which minimizes C_λ

One can show that T_λ is in the produced list of subtrees, hence this algorithm gives the optimal solution.

Final steps:

1. Use cross-validation to select optimal λ^*
2. Retrain on all the data and prune to find T_{λ^*} .

Classification trees (I)

When the output is a categorical variable (say $\mathcal{Y} = \{1, \dots, K\}$) we use the same algorithm above with two important modifications

- For each region R_p we defined the empirical class probabilities:

$$p_{pk} = \frac{1}{m_p} \sum_{\mathbf{x}_i \in R_p} \mathcal{I}[y_i = k]$$

- We classify an input which falls in region p in the class with maximum probability

$$f(\mathbf{x}) = \operatorname{argmax}_{k=1}^K \sum_{p=1}^N p_{pk} \mathcal{I}[\mathbf{x} \in R_p]$$

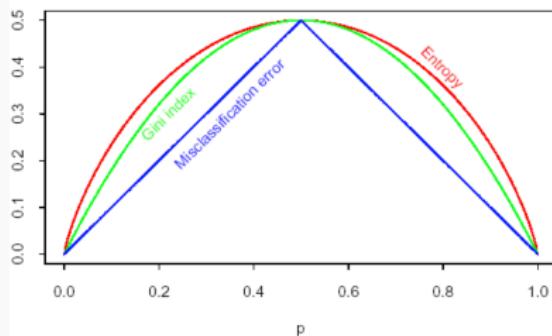
- We use different measures $Q_p(T)$ of **node impurity**.

Classification trees (II)

Node impurity measures:

- Misclassification error:
 $1 - p_{pk}(p)$;
where $k(p) := \operatorname{argmax}_{k=1}^K p_{pk}$

- Gini index:
 $\sum_k p_{pk} (1 - p_{pk})$
- Cross entropy:
 $\sum_k p_{pk} \log \frac{1}{p_{pk}}$

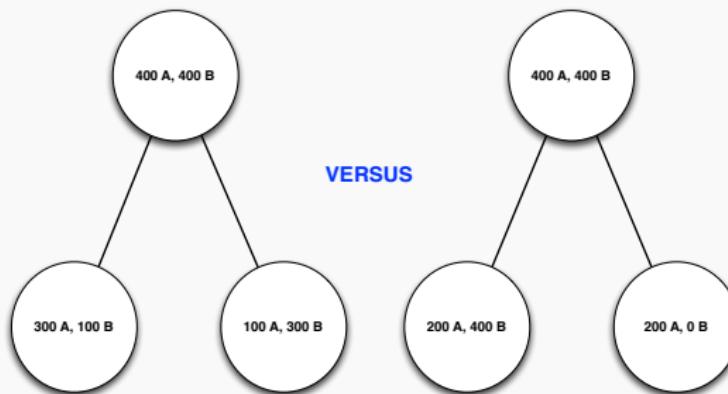


note: cross entropy scaled by 1/2 in figure

- Cross entropy or Gini index are used to grow the tree (they are more sensitive in changes in the node probabilities).
- Misclassification error is often used to prune the tree.

Classification trees (III)

Why use Gini or Entropy as a splitting rule to grow the tree?



Splitting a node with $m_p = 800$ and two classes $\{A, B\}$.

Let's look at the reduction in $\sum_{p=1}^{|T|} m_p Q_p(T)$ for the left versus the right split for the entropy and misclassification impurity measures.

1. Misclassification: Left : $200 = (400 \times 1/4) + (400 \times 1/4)$; Right: $200 = (600 \times 1/3) + (200 \times 0)$
2. Entropy: Left: $649.02 = 400 \times (\frac{1}{4} \log 4 + \frac{3}{4} \log \frac{4}{3}) \times 2$;
Right: $550.98 = 600 \times (\frac{1}{3} \log 3 + \frac{2}{3} \log \frac{3}{2}) + 200 \times 0$

Note that the right split is likely preferable.

Ensemble methods

Part II Bagging and Boosting

Ensemble methods and the wisdom of the crowd – 1

In 1785 Marquis de Condorcet published, *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. In this work he observed among other results that if the probability of a voter being correct is larger than chance then the more voters the greater the likelihood that the majority is correct only increases. In the following we argue this quantitatively.



Marquis de Condorcet

Ensemble methods and the wisdom of the crowd – 2

A motivation:

- A single individual might often be wrong but the crowd majority may often be correct.
- An idealised scenario ...
- Suppose each individual in the crowd $h_1, h_2, \dots, h_{2T+1}$ of size $2T + 1$ predicts the outcome correctly with probability $\frac{1}{2} + \gamma$ independently from one another.

Now consider the “vote” of the crowd.

$$H_T := \text{sign} \left(\sum_{t=1}^{2T+1} h_t \right)$$

What is the probability that H_T will be wrong?

Computing the probability – 1

We have,

$$P(H_T \text{ is wrong}) = \sum_{i=0}^T \binom{2T+1}{i} \left(\frac{1}{2} + \gamma\right)^i \left(\frac{1}{2} - \gamma\right)^{2T+1-i}$$

Hard to evaluate exactly, let use an upper bound.

Chernoff bound

Let X_1, X_2, \dots, X_n be independent random variables. Assume $0 \leq X_i \leq 1$. Let $X := \sum_{i=1}^n X_i$. Let $\mu := E[X] := \sum_{i=1}^n E[X_i]$. Then for all $0 \leq k \leq \mu$,

$$P(X \leq k) \leq \exp\left(-\frac{(\mu - k)^2}{2\mu}\right)$$

Computing the probability – 2

Let $X_1, \dots, X_i, \dots, X_n$ be the Bernoulli random variables. Where $X_i = 1$ if voter i is correct and 0 otherwise. Now, let $k := T$, let $n := 2T + 1$ thus $\mu := (2T + 1)(\frac{1}{2} + \gamma) = T + \frac{1}{2} + 2T\gamma + \gamma$. Now substituting into the bound,

$$\begin{aligned} P(H_t \text{ is wrong}) &\leq \exp\left(-\frac{(\mu - T)^2}{2\mu}\right) \\ &= \exp\left(-\frac{(\frac{1}{2} + 2T\gamma + \gamma)^2}{2(T + \frac{1}{2} + 2T\gamma + \gamma)}\right) \\ &\leq \exp\left(-\frac{4T^2\gamma^2}{5T}\right) \\ &\leq \exp\left(-\frac{4\gamma^2}{5} T\right) \end{aligned}$$

- Bound is crude! But observe the probability that the crowd is wrong exponentially decays to zero.

Bagging

Part II-A

Bagging

Bagging Algorithm

Idea : Reduce the *variance* of a classifier by training many variants of the classifiers and then voting.

Inputs:

1. training data $\mathcal{S} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\} \subset \mathbf{R}^d \times \{-1, 1\}$
2. ensemble size T
3. resample size M
4. classifier function $h_{\mathcal{S}}(\mathbf{x})$

Bagging algorithm:

1. **for** $t = 1$ to T **do**
2. $\mathcal{S}(t) := M$ examples sampled **with replacement** from \mathcal{S}
3. **end for**
4. **return**

$$H(\mathbf{x}) := \text{sign} \left(\sum_{t=1}^T h_{\mathcal{S}(t)}(\mathbf{x}) \right)$$

Conventionally set $M = m$.

How often do examples appear in the bag?

Consider the conventional advice to set $M = m$, then “roughly” how many unique examples from \mathcal{S} are in a “bag” $\mathcal{S}(t)$?

- The probability that **a particular** example does not appear in bag $\mathcal{S}(t)$ is $(1 - \frac{1}{m})^m$.
- Note that

$$\lim_{m \rightarrow \infty} \left(1 - \frac{1}{m}\right)^m = \frac{1}{e} \approx 0.368$$

- Thus **roughly** there will be 63% distinct examples in each $\mathcal{S}(t)$.

Random forests

Idea : Observe for the “wisdom of the crowds” argument to work the predictors need to be independent.

- Thus although decision trees are “high variance” and thus ideal for bagging.
- Let’s go further and try to make the trees more de-correlated.
- When building the tree for each split only check a **subset of size k** features.
- k is usually set to \sqrt{n} or $\log n$.

Useful in practice.

Ensemble methods

Part II-B Boosting

Spam email classification problem

Consider the problem of classifying an email you receive as “good email” or “spam email”

- Different features can be used to represent an email
e.g. the bag of words representation (we may also use additional features to code the text in the subject section of the email etc.)
- Different learning methods (Naive Bayes, SVM, k -NN, CART, etc.) can be used to approximate this task...
- *Key observation:* it is easy to find “*rules of thumb*” that are often correct (say 60% of the time) e.g. “IF ‘business’ occurs in email THEN predict as spam”
- Hard to find highly accurate prediction rules

Weak learners and boosting

Weak learner:

An algorithm which can consistently find classifiers (“rules of thumb”) at least slightly better than random guessing, say better than 55%

Boosting:

A general method of converting rough rules of thumb into a highly accurate prediction rule (classifier)

Boosting algorithm

- Devise a computer program for deriving rough rules of thumb
- Choose rules of thumb to fit a subset of example
- Repeat T times
- Combine the classifiers by weighted majority vote

Key steps are:

- how do we choose the subset of examples at each round?
concentrate on **hardest examples** (those most often
misclassified by previous classifiers)
- how do we combine the weak learners? by **weighted majority**

Adaboost Glossary

- $D_t(i)$: the weight on example i at time t where $\sum_{i=1}^m D_t(i) = 1$.
- α_t : the weight on weak learner t where $\alpha_t \in \mathbf{R}$.
- $h_t(\cdot)$: is the weak learner “generated” at time t which is a function from $\mathbf{R}^d \rightarrow \{-1, 1\}$.
- $f(\cdot)$: $f(\mathbf{x}) := \sum_{i=1}^T \alpha_i h_i(\mathbf{x})$.
- $H(\cdot)$: $H(\mathbf{x}) := \text{sign}(f(\mathbf{x}))$. (Final classifier output by adaboost)
- ϵ_t : “weighted error of weak learner $h_t(\cdot)$ at time t ”

$$\epsilon_t := \sum_{i=1}^m D_t(i) \mathcal{I}[h_t(\mathbf{x}_i) \neq y_i]$$

- **weak learner generator**: given input

$$D_t(1), \dots, D_t(m), (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$$

outputs a weaker learner $h_t(\cdot)$ such that $\epsilon_t < \frac{1}{2}$.

Adaboost algorithm

Input: training data $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$

Initialise: $D_1(1) = \dots = D_1(m) = \frac{1}{m}$

For $t = 1, \dots, T$:

1. fit a classifier $h_t : \mathbf{R}^d \rightarrow \{-1, 1\}$ using distribution D_t
2. choose $\alpha_t \in \mathbf{R}$ (see (1) in following slide)
3. update: for each $i \in \{1, \dots, m\}$

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}}{Z_t} \quad (*)$$

Where Z_t is a normalization factor (so as to ensure that D_{t+1} is a distribution, i.e. $\sum_{i=1}^m D_{t+1}(i) = 1$.)

Return: the final classifier

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right)$$

Some remarks

Terminology:

- The basic idea in Adaboost is to maintain a distribution D on the training set and iteratively train a weak learner on it
- At each round larger weights are assigned to hard examples, hence the weak learner will focus mostly on those examples

Let ϵ_t be the *weighted* training error of classifier t :

$$\epsilon_t = \sum_{i=1}^m D_t(i) \mathcal{I}[h_t(\mathbf{x}_i) \neq y_i]$$

We will justify later the following choice for α_t

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t} \quad (1)$$

Weight by majority

The final classifier is a weighted majority vote of the T base classifiers where α_t is the weight assigned to h_t

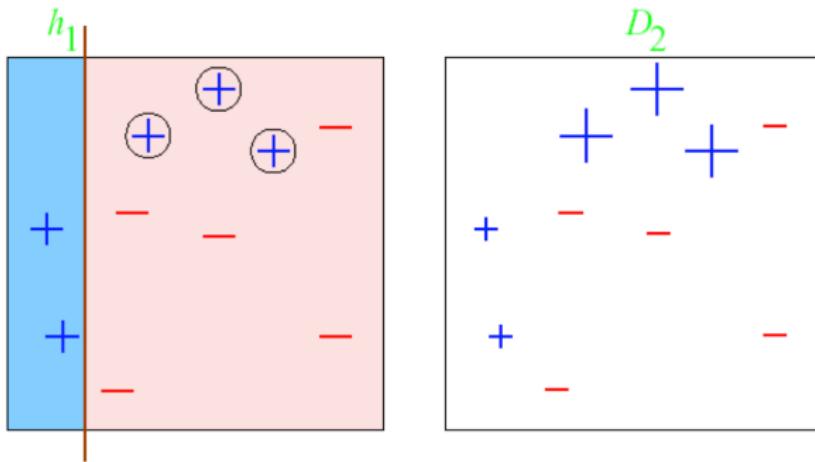
$$f(\mathbf{x}) = \sum_{t=1}^T \alpha_t h_t$$

Typically $\epsilon_t \leq 0.5$ hence $\alpha_t \geq 0$ (this is always the case if h_t and $-h_t$ are both in the set of weak learners, e.g. for decision stumps)

Thus, f is essentially a linear combination of the h_t with weights controlled by the training error

Example (round 1)

Let's discuss a simple example where the weak learners are decision stumps (vertical or horizontal lines, i.e. trees with only two leaves)



$$\epsilon_1 = 0.30$$

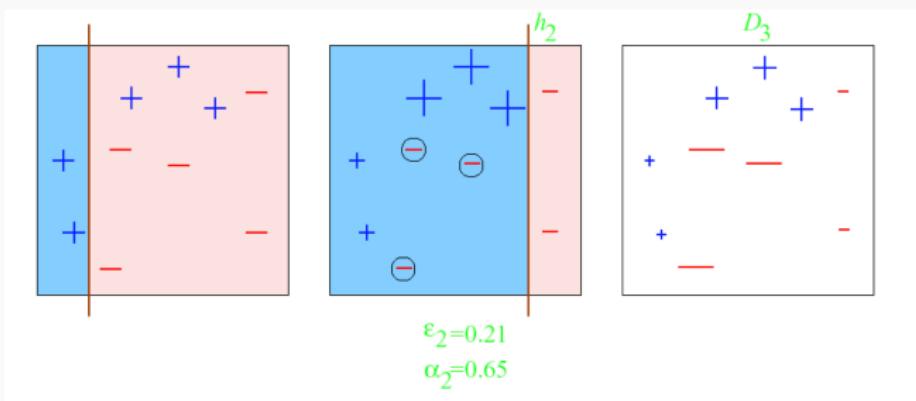
$$\alpha_1 = 0.42$$

1. The 1st plot illustrates the misclassified examples
2. The 2nd plot illustrates how the difficult gain more weight

(Thanks to Rob Schapire for providing the figures for this example)

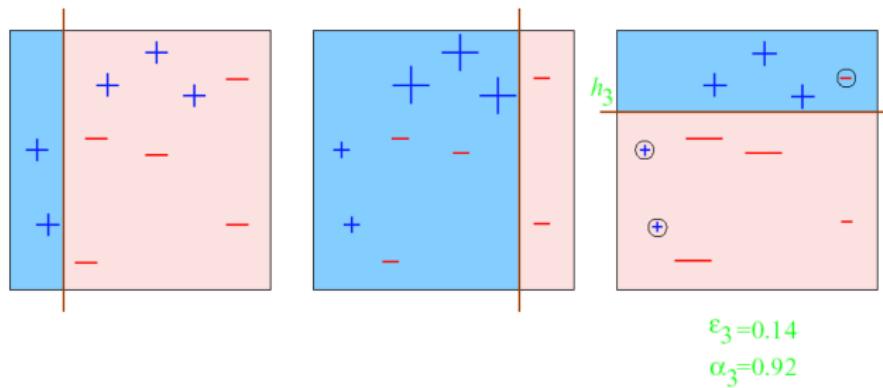
Example (round 2)

- The second classifier concentrates more on the difficult examples.
- Examples are then re-weighted according to this classifier



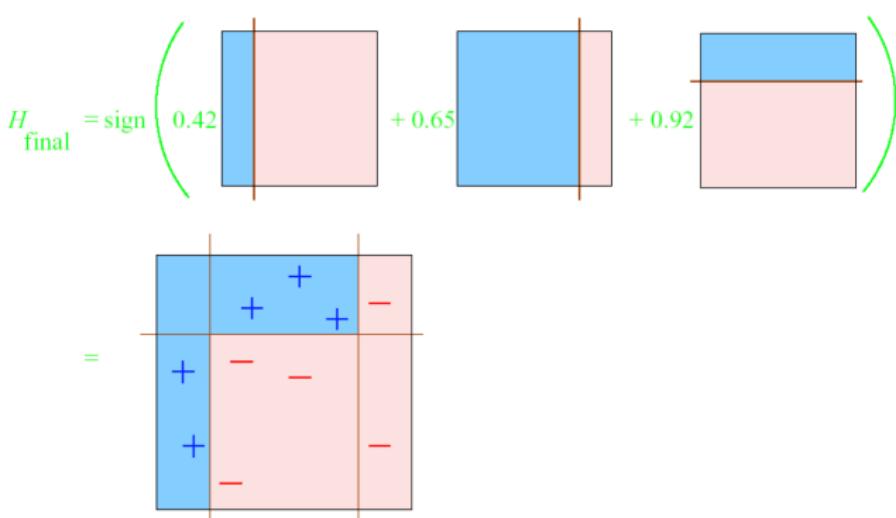
Example (round 3)

In this example the set of weak learners can be assumed to be finite. There are roughly $4m$ weak learners/decision stumps corresponding to choosing either the horizontal or vertical coordinate and splitting between any two points.



Example (final round)

The final classifier has zero training error!



Boosting Converges

Theorem

Given a training set $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$ and assume that each iteration of Adaboost a weaker learner returns a hypothesis with weighted error $\epsilon_t \leq \frac{1}{2} - \gamma$. Then the training error of the output hypothesis of Adaboost is at most

$$\frac{1}{m} \sum_{i=1}^m \mathcal{I}[H(\mathbf{x}_i) \neq y_i] \leq \exp(-2\gamma^2 T)$$

We argue this over the next five slides.

Analysis of the training error (I)

The training error of the boosting algorithm is bounded as

$$\frac{1}{m} \sum_{i=1}^m \mathcal{I}[H(\mathbf{x}_i) \neq y_i] \leq \frac{1}{m} \sum_{i=1}^m e^{-y_i f(\mathbf{x}_i)} = \prod_{t=1}^T Z_t$$

where we have defined

$$f := \sum_t \alpha_t h_t \quad (\text{so that } H(\mathbf{x}) = \text{sign}(f(\mathbf{x})))$$

1. The inequality follows from:

$$H(\mathbf{x}_i) \neq y_i \Rightarrow e^{-y_i f(\mathbf{x}_i)} \geq 1$$

2. The equality follows from the recursive definition of D_t

Analysis of the training error (II)

Expanding on point 2:

$$D_{T+1}(i) = \frac{1}{m} \frac{\prod_{t=1}^T e^{-\alpha_t y_i h_t(\mathbf{x}_i)}}{\prod_{t=1}^T Z_t}$$

and expanding

$$\begin{aligned} \frac{1}{m} \sum_{i=1}^m e^{-y_i f(\mathbf{x}_i)} &= \frac{1}{m} \sum_{i=1}^m e^{-y_i \sum_{t=1}^T \alpha_t h_t(\mathbf{x}_i)} \\ &= \frac{1}{m} \sum_{i=1}^m \prod_{t=1}^T e^{-y_i \alpha_t h_t(\mathbf{x}_i)} \\ &= \sum_{i=1}^m D_{T+1}(i) \prod_{t=1}^T Z_t \\ &= \prod_{t=1}^T Z_t \end{aligned} \quad \square$$

Analysis of the training error (III)

The previous bound suggests that if at each iteration we choose α_t and h_t by minimizing Z_t the final training error of H will be reduced most rapidly

Since h_t maps to $\{-1, 1\}$ (they are binary classifiers) then Z_t is minimized by the choice of equation (1) above

$$\text{Recall (1): } \alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

We show this next

Analysis of the training error (IV)

From formula (*) we have

$$Z_t = \sum_i D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}$$

Using the fact that h_t returns binary values we also have that

$$\begin{aligned} Z_t &= e^{\alpha_t} \sum_{i: y_i \neq h_t(\mathbf{x}_i)} D_t(i) + e^{-\alpha_t} \sum_{i: y_i = h_t(\mathbf{x}_i)} D_t(i) \\ &= \epsilon_t e^{\alpha_t} + (1 - \epsilon_t) e^{-\alpha_t} \end{aligned}$$

Equation (1) now follows by solving $\frac{dZ_t}{d\alpha_t} = 0$

Analysis of the training error (V)

Placing $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$ in the above formula for Z_t we obtain

$$Z_t = \epsilon_t e^{\alpha_t} + (1 - \epsilon_t) e^{-\alpha_t} = 2\sqrt{\epsilon_t(1 - \epsilon_t)} = \sqrt{1 - 4\gamma_t^2}$$

where $\gamma_t = 1/2 - \epsilon_t$. Hence we have the following bound for the training error

$$\frac{1}{m} \sum_{i=1}^m \mathcal{I}[H(\mathbf{x}_i) \neq y_i] \leq \prod_t Z_t = \prod_t \sqrt{1 - 4\gamma_t^2} \leq e^{-2 \sum_t \gamma_t^2}$$

In the final inequality we used the fact $1 - x \leq e^{-x}$. Thus, if each weak classifier is slightly better than random guessing (if $\gamma_t \geq \gamma > 0$) the training error drops **exponentially fast**

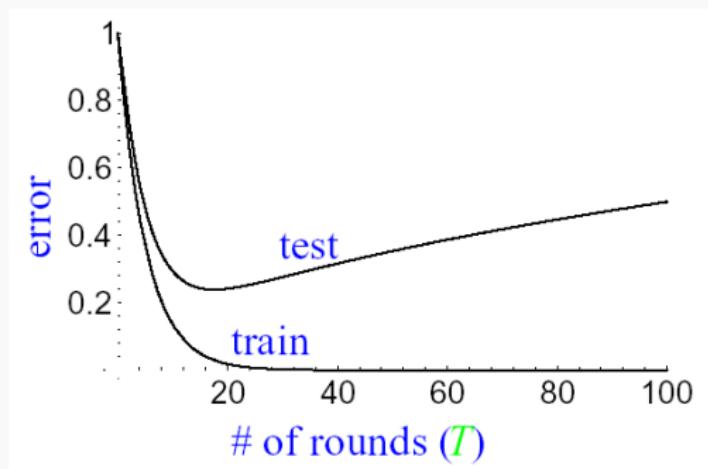
$$\text{training error} \leq e^{-2T\gamma^2}$$

Generalization error (I)

How do we expect training and test error of boosting to depend on T ?

At first thought, we might expect that if T is too large that we'd overfit.

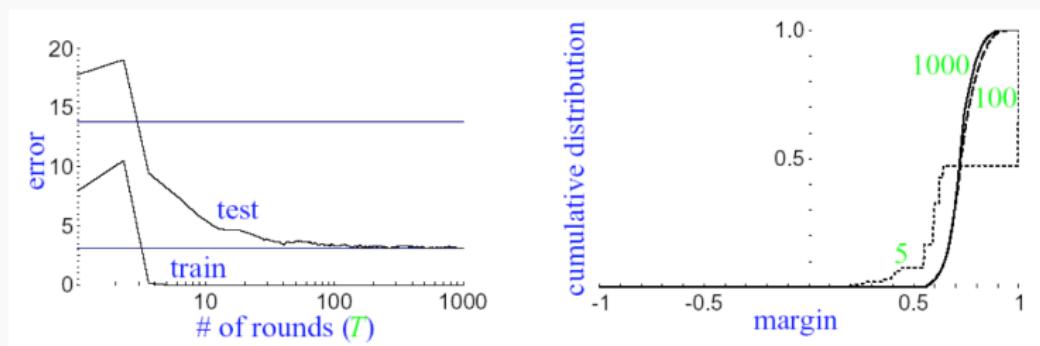
Thus possibly a curve like the following.



(Figures from Schapire *et. al*)

Generalization error (II)

Typically, the test error keeps decreasing even when the training error is zero! (eventually it will increase but may take many more iterations to do so)



However the **margin distribution** “concentrates” as well which explains why test error improves.

The margin of point i is simply the quantity $\text{margin}_i = y_i f(\mathbf{x}_i)$ where we assume that $\sum_t \alpha_{t=1}^T = 1$ (just normalize the weights after the last iteration of boosting). The margin distribution is the empirical distribution of the margin

One can show that with high probability (say 99%)

$$\text{generalization error} \leq \Pr_{\text{emp}}(\text{margin} \leq \theta) + O\left(\frac{\sqrt{\text{vc}/m}}{\theta}\right)$$

where m is the number of training points, vc is the VC-dimension of the set of weak learners and \Pr_{emp} denotes empirical probability of the margin (like the training error this converges exponentially fast to zero)

Note: that for the hypothesis class of decision stumps in \mathbf{R}^d that

$$\text{vc} \leq 2(\log d + 1)$$

Motivating Adaboost

A motivation for adaboost

Additive models and boosting (I)

Boosting can be seen as a greedy way to solve the problem

$$\min \left\{ \sum_{i=1}^m V \left(y_i, \sum_{t=1}^T \alpha_t h_t(\mathbf{x}_i) \right) : \alpha_1, \dots, \alpha_T \in \mathbf{R}^T, h_1, \dots, h_T \in \mathcal{H}^T \right\}$$

where \mathcal{H} is the hypothesis class which contain the “weak” learners.
And the loss function is exponential loss as we shall see

$$V(y, \hat{y}) = \exp(-y\hat{y}).$$

At each iteration a new basis function is added to the current basis expansion $f^{(t-1)} = \sum_{s=1}^{t-1} \alpha_s h_s$

$$(\alpha_t, h_t) = \operatorname{argmin}_{\alpha_t, h_t} \sum_{i=1}^m V(y_i, f^{(t-1)}(\mathbf{x}_i) + \alpha_t h_t(\mathbf{x}_i))$$

- This is unlike in CART, where at each iteration previous basis function coefficients are re-adjusted.

Additive models and boosting (II)

$$\min_{\alpha_t, h_t} \sum_{i=1}^m V(y_i, f^{(t-1)}(\mathbf{x}_i) + \alpha_t h_t(\mathbf{x}_i))$$

In the statistical literature, this type of algorithm is called *forward stagewise additive model*.

To derive adaboost, substitute in $V(y, \hat{y}) = \exp(-y\hat{y})$ and minimise.
Thus,

$$\min_{\alpha_t, h_t} \sum_{i=1}^m e^{-y_i(f^{(t-1)}(\mathbf{x}_i) + \alpha_t h_t(\mathbf{x}_i))}$$

Define $\mathcal{D}_t(i) := e^{-y_i f^{(t-1)}(\mathbf{x}_i)}$ and hence we have

$$\min_{\alpha_t, h_t} \sum_{i=1}^m \mathcal{D}_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}$$

Additive models and boosting (III)

Observe that (w.l.o.g.) that the minimum of the below with respect to h_t

$$\min_{\alpha_t, h_t} \sum_{i=1}^m \mathcal{D}_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}$$

is independent of **positive** α_t , i.e., rewriting the above as,

$$\min_{\alpha_t, h_t} \left(e^{\alpha_t} \sum_{i: y_i \neq h_t(\mathbf{x}_i)} \mathcal{D}_t(i) + e^{-\alpha_t} \sum_{i: y_i = h_t(\mathbf{x}_i)} \mathcal{D}_t(i) \right)$$

and then

$$\min_{\alpha_t, h_t} \left((e^{\alpha_t} - e^{-\alpha_t}) \sum_{i=1}^m \mathcal{D}_t(i) \mathcal{I}[y_i \neq h_t(\mathbf{x}_i)] + e^{-\alpha_t} \sum_{i=1}^m \mathcal{D}_t(i) \right)$$

Now we can see that we have derived adaboost! As

1. The h_t that minimises the above minimises the misclassification error weighted by \mathcal{D}_t
2. The \mathcal{D}_t defined here is proportional to the adaboost " D_t " see *Analysis of training error (II)*
3. And now the minimisation we perform to find α_t is the same as in *Analysis of the training error (IV)*

Why the exponential loss for classification? (I)

Recall the following typical set-up for classification. We are given a family of hypothesis \mathcal{F} and wish to find a $f \in \mathcal{F}$ so that we predict well on future data. One approach is to find an f that minimizes the loss ($\lambda = 0$) on the empirical data or regularized loss ($\lambda > 0$), thus solving

$$\min_{f \in \mathcal{F}} \sum_{i=1}^m V(y_i, f(\mathbf{x}_i)) + \lambda \text{complexity}(f)$$

Problems: In classification as opposed to regression we face two (related) problems:

1. The misclassification loss is not smooth hence difficult to optimize
2. To make the class the function \mathcal{F} both **rich** and **smooth** typically we choose the class so that the functions f map to \mathbf{R} rather than $\{-1, 1\}$ and then predict with $\text{sign}(f)$

Thus we are motivated for the purpose of finding a hypothesis $f \in \mathcal{F}$ to choose V to be different from the misclassification loss.

Why the exponential loss for classification? (II)

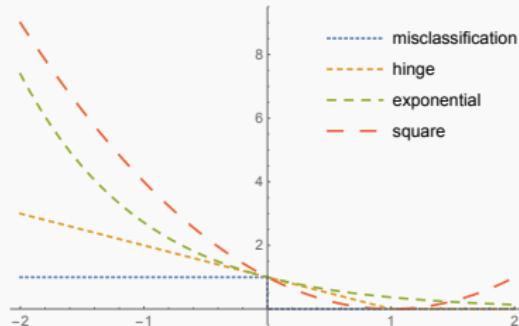
Consider the following typical loss functions:

$$V_{\text{mc}}(y, \hat{y}) = \mathcal{I}[y = \text{sign}(\hat{y})]$$

$$V_{\text{hinge}}(y, \hat{y}) = \max(0, 1 - y\hat{y})$$

$$V_{\text{exp}}(y, \hat{y}) = \exp(-y\hat{y})$$

$$V_{\text{sq}}(y, \hat{y}) = (y - \hat{y})^2$$



$$V_{\text{mc}}(1, \hat{y}), V_{\text{hinge}}(1, \hat{y}), V_{\text{exp}}(1, \hat{y}), \text{ and } V_{\text{sq}}(1, \hat{y})$$

The margin of a prediction \hat{y} is $y\hat{y}$ for $y \in \{-1, 1\}$.

1. Misclassification is not continuous
2. Square loss unnecessarily punishes predictions with increasing positive margin
3. Hinge loss has the nice property that it punishes negative margins but not positive margins
4. Hinge loss is not differentiable everywhere however.
5. Exponential loss punishes negative margins and promotes large positive margins (secondarily).

Boosting Summary

- Given a *weak* learner oracle
- Boosting finds a linear combination of weak hypotheses with arbitrarily small *training* error
- With certain assumptions this implies that the *test* error is bounded with high probability
- Boosting may be motivated as a *forward stagewise additive model* with exponential loss

Decision trees and ensembles summary

- Decision tree-based methods are useful in practice
- They provide “human-interpretable” models
- A strong advantage is that they work on **ordinal** data. Less feature cleaning needed.
- Ensemble methods are often used on top of trees
- Bagging tends to use full trees
- Boosting often uses only decision stumps (single split trees)
- In the best cases (problem dependent)
 - Bagging tends to reduce variance
 - Boosting tends to reduce bias
- Boosting generally has more problems on very noisy data

Suggested Readings

The lecture notes are based on *The Elements of Statistical Learning* ..., Chapters 9.2 and 10.

Also recommended *The Boosting Approach to Machine Learning An Overview*, Schapire; *A decision-theoretic generalization of on-line learning and application to boosting*, Freund and Schapire, *Boosting the margin: A new explanation for the effectiveness of voting methods*, Schapire, Freund, Bartlett, and Lee

For a compact presentation of boosting and decision trees and further theory see Chapters 10 & 18 of *Understanding Machine Learning from Theory to Algorithms*

Finally for a tutorial on decision forests, see particularly Chapters 1-3. Includes many graphics and a nice example for Human Body Tracking with the Microsoft Kinect. See *Decision Forests for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning*, MSR technical report TR-2011-114.

Going Deeper ...

- *Decision trees as partitioning machines to characterize their generalization properties.* Bounds the VC-dimension of decision stumps and binary trees and introduces a pruning algorithm that performs better than CART on a number of datasets.
- *From global to local MDI variable importances for random forests and when they are Shapley values.* Random forest are often used to evaluate the “relative importance” of variables. See this paper for recent work on the subject.

Problems – 1

- Given the dataset

$$\{((1, 1), 9), ((1, 2), -4), ((1, 3), 2), ((2, 2), 4), ((2, 3), 2)\}$$

find the regression tree corresponding to the greedy recursive partitioning procedure with respect to square error.

- 1.1 Draw the recursive partition.
1.2 Draw the tree representation.
- Explain how a random forest is constructed. What is the motivation for the tree construction method?
- Both bagging and boosting produce predictions by creating an ensemble of functions. Explain how these ensembles differ (you do not need to describe the methods for arriving at the ensembles just how structurally the ensembles will differ).

Problems – 2

1. The Adaboost initilisation and update.

- 1.1 Explain how Adaboost chooses its initial weighting $D_0(i)$ of the examples

$$S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$$

- 1.2 Explain how it *updates* the distribution D_{t-1} after choosing a weak learner h_t with coefficient α_t at stage t including the normalising constant $Z(t)$.
 - 1.3 Give an intuitive justification for this update.
2. Explain how Adaboost uses the distribution D_t that weights the examples $((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m))$ to choose a weak learner from a set H at stage t .
3. Using the result of part (1.2) obtain a bound for

$$\frac{1}{m} \sum_{i=1}^m I[\text{sign}(f_T(\mathbf{x}_i)) \neq y_i] \leq \frac{1}{m} \sum_{i=1}^m \exp(-y_i f_T(\mathbf{x}_i)),$$

in terms of the normalisation constants $Z(1), \dots, Z(T)$, where $I[x]$ is the indicator function with value 1 when x is true and 0 otherwise, and $f_T(\mathbf{x}) = \sum_{t=1}^T \alpha_t h_t(\mathbf{x})$.

Problems – 3

1. A data set $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$ is *generic* iff $\mathbf{x}_i = \mathbf{x}_j \Rightarrow y_i = y_j$. A set of functions \mathcal{H} is called *weakly-universal* if for every generic data set and weighting of the data there exists a function $h \in \mathcal{H}$ with “weighted error” < 0.5. [subtle]

- A positive decision stump $h_{i,z} : \mathbf{R}^n \rightarrow \{-1, 1\}$ is defined as

$$h_{i,z}(\mathbf{x}) := \begin{cases} 1 & \text{if } x_i \leq z \\ -1 & \text{if } x_i > z \end{cases}.$$

The set of all decision stumps is defined as

$$\mathcal{H}_{\text{ds}} := \{s h_{i,z} : s \in \{-1, 1\}, i \in \{1, 2, \dots, n\}, z \in \mathbf{R}\}.$$

Is \mathcal{H}_{ds} weakly-universal? Provide an argument supporting your answer.

- Let,

$$S := \{a, b, aa, ab, ba, bb, aaa, \dots\}$$

denote the set of all strings of non-zero finite length over $\{a, b\}$. A string x is a *substring* of y , denoted as $x \sqsubseteq y$ iff x is contained as a consecutive sequence of characters in y . Thus $a \sqsubseteq abba$, $ab \sqsubseteq abba$, $abba \sqsubseteq abba$ but $aba \not\sqsubseteq abba$ and $abbaa \not\sqsubseteq abba$. A positive substring-match function $g : S \rightarrow \{-1, 1\}$ is defined as,

$$g_z(x) := \begin{cases} 1 & \text{if } x \sqsubseteq z \\ -1 & \text{if } x \not\sqsubseteq z \end{cases}.$$

The set of all substring-match functions is denoted as

$$\mathcal{H}_{\text{ss}} := \{s g_z : s \in \{-1, 1\}, z \in S\}.$$

Is \mathcal{H}_{ss} weakly-universal? Provide an argument supporting your answer.

5. Online learning I

COMP0078: Supervised Learning

Mark Herbster

1 November 2021

University College London
Department of Computer Science
SL-onlinelearning21v1

Acknowledgments and References

Thanks

Thanks to Yoav Freund and Claudio Gentile for some of the slides.

A general reference for online learning

- Nicolò Cesa-Bianchi and Gábor Lugosi, *Prediction, learning, and games*

A more technical recent reference

- Shai Shalev-Shwartz, *Online Learning and Online Convex Optimization*

See final slide for more references.

Batch versus Online learning

Batch

Model: There exists **training** data set (sampled **IID**)

Aim: To build a classifier from the training data that predicts well on new data (*from same distribution*)

Evaluation metric: *Generalization error*

Online

Model: There exists an **online sequence** of data (*usually no distributional assumptions*)

Aim: To sequentially predict and update a classifier to predict well on the sequence (i.e. there is no training and test set distinction)

Evaluation metric: *Cumulative error*

Note

There are a variety of models for online learning. Here we focus on the so-called *worst-case* model. Alternately distributional assumption may be made on the data sequence. Also sometimes the phrase "online learning" is used to refer to "online optimisation" that is to use online learning type algorithms as a *training* method for a batch classifier.

Why online learning?

Pragmatically

- “Often” fast algorithms
- “Often” small memory footprint
- “Often” no “statistical” assumptions required e.g. IID-ness
- As a *training* method for “**BIG DATA**” batch classifiers

Theoretically (learning performance guarantees)

- Non-asymptotic
- No statistical assumptions
- There exist techniques to convert *cumulative error* guarantees to *generalisation error* guarantees

Today

Our focus today is on three foundational online “hypotheses” classes.

- Learning with experts
 1. Halving algorithm
 2. Weighted Majority algorithm
 3. Refining and generalising the experts model
- Learning linear classifiers
 1. Perceptron
 2. Winnow
 3. Case study: Using perceptron and winnow to learn Disjunctions and DNF
- Learning with sequences of experts

Experts

Part I Learning with Expert Advice

On-Line Learning with expert advice (1) [V90,LW94, HKW98]

Model: There exists an **online sequence** of data

$$S = (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m) \in \{0, 1\}^n \times \{0, 1\}.$$

Interpretation: The vector \mathbf{x}_t is the set of predictions of n experts about an outcome y_t . Where expert i predicts $x_{t,i} \in \{0, 1\}$ at time t . Each expert at time t is aiming to predict y_t . What is an “expert”? These may be for example human experts or the predictions of some algorithm.

	experts						
	E_1	E_2	E_3	E_n			
day 1	1	1	0	0	0	1	1
day 2	1	0	1	0	1	0	1
day 3	0	1	1	1	1	1	0
day t	$x_{t,1}$	$x_{t,2}$	$x_{t,3}$	$x_{t,n}$	\hat{y}_t	y_t	$ y_t - \hat{y}_t $

Goal: A “Master” algorithm to combine the predictions \mathbf{x}_t of the n experts (based on past perf.) to predict \hat{y}_t an estimate of y_t .

On-Line Learning with experts (2)

Protocol of the Master Algorithm

For $t = 1$ To m Do

Get instance $\mathbf{x}_t \in \{0, 1\}^n$

Predict $\hat{y}_t \in \{0, 1\}$

Get label $y_t \in \{0, 1\}$

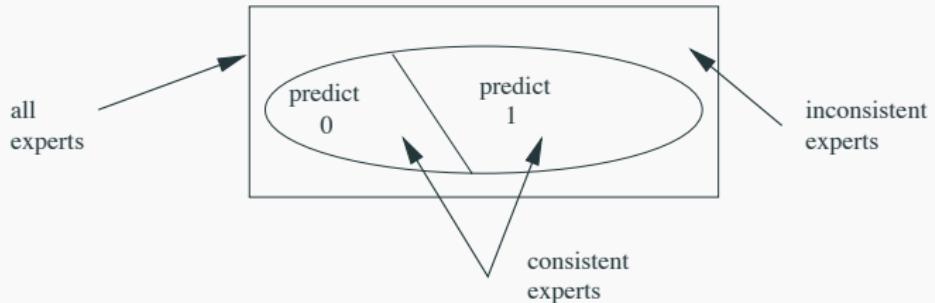
Incur loss (mistakes) $|y_t - \hat{y}_t|$

Evaluation metric: The loss (mistakes) of Master Algorithm A on sequence S is just

$$L_A(S) := \sum_{t=1}^m |y_t - \hat{y}_t|$$

Our Goal: Design master algorithms with “small loss”.

A Solution : Halving Algorithm



- Predicts with majority
- If mistake is made then number of **consistent experts** is (at least) **halved**

A run of the Halving Algorithm

E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	<i>majo rity</i>	<i>true label</i>	loss
1	1	0	0	1	1	0	0	1	0	1
x	x	0	1	x	x	1	1	1	1	0
x	x	x	1	x	x	0	0	0	1	1
x	x	x	↑	x	x	x	x			
consistent										

Observation: For any sequence with a **consistent** expert Halving Algorithm makes $\leq \log_2 n$ mistakes.

Exercise: Prove this!

What if no expert is consistent?

Notation

- Recall $L_A(S) := \sum_{t=1}^m |y_t - \hat{y}_t|$ is the loss of algorithm A on S
- Let

$$L_i(S) := \sum_{t=1}^m |y_t - x_{t,i}|$$

be the loss of i -th expert E_i

Aim

Bounds of the form:

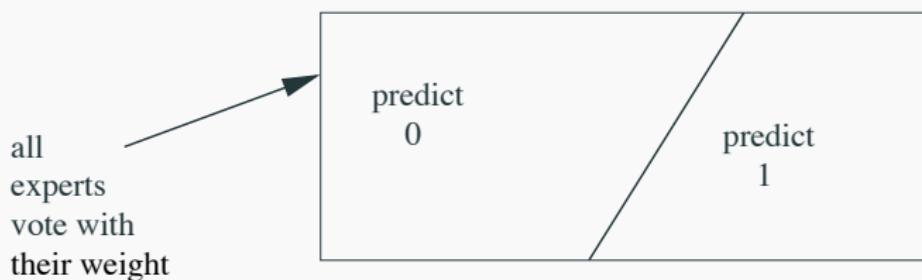
$$\forall S : L_A(S) \leq a \min_i L_i(S) + b \log(n)$$

where a, b are “small” constants

Comment: These are known as “Regret”, “Relative” or “Worst-case” loss bounds, i.e., the bounds on the loss of algorithm are “relative to” loss of best expert and they hold even in the “worst-case.”

Can't wipe out experts!

One weight per expert



- Predicts with larger side
- Weights of wrong experts are multiplied by $\beta \in [0, 1)$

Number of mistakes of the WM algorithm

$M = \#$ mistakes of master algorithm

$$M_{t,i} = \# \text{ mistakes of expert } E_i \text{ at the start of trial } t$$

$$M_i = M_{m+1,i} \quad = \quad \# \text{ of total mistakes of expert } E_i$$

$$w_{t,i} = \beta^{M_{t,i}} \text{ weight of } E_i \text{ at beginning of trial } t \quad (w_{1,i} = 1)$$

$$W_t = \sum_{i=1}^n w_{t,i} \text{ total weight at trial } t$$

$$\text{Minority: } \leq \frac{1}{2} W_t \quad \text{Majority: } \geq \frac{1}{2} W_t$$

If no mistake then minority multiplied by β :

$$W_{t+1} \leq \textcolor{red}{1} W_t$$

Else If mistake then majority multiplied by β :

$$= \frac{1 + \beta}{2} W_t$$

Number of mistakes of the WM algorithm – Continued-1

Hence

$$W_{m+1} \underset{\substack{\text{total final} \\ \text{weight}}}{\leq} \left(\frac{1+\beta}{2} \right)^M W_1$$

$$W_{m+1} = \sum_{j=1}^n w_{m+1,j} = \sum_{j=1}^n \beta^{M_j} \geq \beta^{M_i}$$

We get:

$$\left(\frac{1+\beta}{2} \right)^M \underbrace{W_1}_n \geq \beta^{M_i}$$

Number of mistakes of the WM algorithm – Continued-2

Solving for M :

$$M \leq \frac{\ln \frac{1}{\beta}}{\ln \frac{2}{1+\beta}} M_i + \frac{1}{\ln \frac{2}{1+\beta}} \ln n$$

$$M \leq \underbrace{\frac{2.63}{\beta}}_{a} \min_i M_i + \underbrace{2.63 \ln n}_{b}$$

For **all** sequences, loss of master algorithm is **comparable to** loss of best expert. Thus the terminology **Relative** loss bound.

Refining and generalising the experts model – 1

More generally we would like to obtain *regret* bounds for arbitrary loss functions $L : \mathcal{Y} \times \hat{\mathcal{Y}} \rightarrow [0, +\infty]$. Making our notion of regret more precise we would like guarantees of the form,

$$L_A(S) - \min_{i \in [n]} L_i(S) \leq o(m),$$

where the right-hand side is termed regret since it is how much we “regret” predicting with the algorithm as opposed to the optimal predictor on the data sequence. Here $o(m)$ denotes some function sublinear in m (the number of examples in S) and possibly dependent on other parameters.

Therefore since

$$\frac{L_A(S) - \min_{i \in [n]} L_i(S)}{m} \leq \frac{o(m)}{m},$$

and thus as $m \rightarrow \infty$ we have

$$\frac{L_A(S)}{m} \leq \frac{\min_{i \in [n]} L_i(S)}{m},$$

thus in the limit the mean asymptotic loss of our algorithm is bounded by the mean asymptotic loss of the best expert.

Refining and generalising the experts model – 2

In the following we will show two examples of regret bounds generalising the analysis of the weighted majority algorithm.

1. For a loss function $L : \{0, 1\} \times [0, 1] \rightarrow [0, +\infty]$ the *entropic loss*
$$L(y, \hat{y}) := y \ln \frac{y}{\hat{y}} + (1 - y) \ln \frac{1-y}{1-\hat{y}}$$
2. For an arbitrary bounded loss function $L : \mathcal{Y} \times \hat{\mathcal{Y}} \rightarrow [0, B]$.

For the first the regret will be the small constant $\log(n)$ for the second the regret will be $O(\sqrt{m \log n})$.

A regret bound for the entropic (log) loss

Unlike in the case of the weighted majority we will now allow predictions in $[0, 1]$ rather than just $\{0, 1\}$ and rather prediction with the “majority” we will predict with the *weighted average*.

One weight per expert:

$$w_{t,i} = \beta^{L_{t,i}} = e^{-\eta L_{t,i}}$$

where $L_{t,i}$ is cumulative loss of E_i before trial t
and η is a positive learning rate

Master predicts with the **weighted average (dot product)**

$$\begin{aligned} v_{t,i} &= \frac{w_{t,i}}{\sum_{i=1}^n w_{t,i}} \quad \text{normalized weights} \\ \hat{y}_t &= \sum_{i=1}^n v_{t,i} x_{t,i} = \mathbf{v}_t \cdot \mathbf{x}_t \end{aligned}$$

where $x_{t,i}$ is the prediction of E_i in trial t

Set the initial weights $\mathbf{v}_1 := \mathbf{w}_1 := (\frac{1}{n}, \dots, \frac{1}{n})$.

The WA Algorithm – Summary

WA Algorithm

Initialise : $\mathbf{v}_1 := \mathbf{w}_1 := (\frac{1}{n}, \dots, \frac{1}{n})$, $L_{\text{WA}} := 0$, $\mathbf{L} := \mathbf{0}$,

Select: $\eta \in (0, \infty)$, Loss function $L : (\cdot, \cdot)$.

For $t = 1$ To m Do

 Receive instance $\mathbf{x}_t \in [0, 1]^n$

 Predict $\hat{y}_t := \mathbf{v}_t \cdot \mathbf{x}_t$

 Receive label $y_t \in [0, 1]$

 Incur loss $L_{\text{WA}} := L_{\text{WA}} + L(y_t, \hat{y}_t)$, $L_i := L_i + L(y_t, x_{t,i})$ ($i \in [n]$)

 Update Weights $v_{t+1,i} := \frac{v_{t,i} e^{-\eta L(y_t, x_{t,i})}}{\sum_{j=1}^n v_{t,j} e^{-\eta L(y_t, x_{t,j})}}$ for $i \in [n]$.

Weighted Average Algorithm - Theorem

Theorem

For all sequences of examples

$$S = (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m) \in [0, 1]^n \times [0, 1]$$

the regret of the *weighted average* WA algorithm is

$$L_{\text{WA}}(S) - \min_i L_i(S) \leq \underbrace{1/\eta}_{b} \ln(n)$$

with square and entropic loss for $\eta = 1/2$ and $\eta = 1$ respectively.

Constant b as dependent on loss function

Loss		$b = 1/\eta$
entropic	$L_{\text{en}}(y, \hat{y}) = y \ln \frac{y}{\hat{y}} + (1-y) \ln \frac{1-y}{1-\hat{y}}$	1
square	$L_{\text{sq}}(y, \hat{y}) = (y - \hat{y})^2$	2

- For simplicity, we will prove only for entropic loss when $\mathcal{Y} := \{0, 1\}$ and $\hat{\mathcal{Y}} := [0, 1]$. The result holds for many loss functions (sufficient smoothness and convexity with different η and b). See [KW99] for proof.

Weighted Average Algorithm - Proof

Notation: $\Delta_n := \{\mathbf{x} \in [0, 1]^n : \sum_{i=1}^n x_i = 1\}$. Let $d : \Delta_n \times \Delta_n \rightarrow [0, \infty]$ be the rel. entropy $d(\mathbf{u}, \mathbf{v}) := \sum_{i=1}^n u_i \ln \frac{u_i}{v_i}$. Note: $L_{\text{en}}(y, \hat{y}) = d((y, 1-y), (\hat{y}, 1-\hat{y}))$.

Proof – 1

We first prove the following “progress versus regret” equality.

$$L_{\text{en}}(y_t, \hat{y}_t) - \sum_{i=1}^n u_i L_{\text{en}}(y_t, x_{t,i}) = d(\mathbf{u}, \mathbf{v}_t) - d(\mathbf{u}, \mathbf{v}_{t+1}) \text{ for all } \mathbf{u} \in \Delta_n. \quad (1)$$

Observe that

$$d(\mathbf{u}, \mathbf{v}_t) - d(\mathbf{u}, \mathbf{v}_{t+1}) = \sum_{i=1}^n u_i \ln \frac{v_{t+1,i}}{v_{t,i}}$$

If $y_t = 1$ then (using $L_{\text{en}}(1, x) = -\ln x$)

$$\begin{aligned} \sum_{i=1}^n u_i \ln \frac{v_{t+1,i}}{v_{t,i}} &= \sum_{i=1}^n u_i \ln \frac{\frac{v_{t,i} e^{-L_{\text{en}}(1, x_{t,i})}}{\sum_{i=1}^n v_{t,i} e^{-L_{\text{en}}(1, x_{t,i})}}}{v_{t,i}} = \sum_{i=1}^n u_i \ln \frac{\frac{v_{t,i} x_{t,i}}{\sum_{i=1}^n v_{t,i} x_{t,i}}}{v_{t,i}} = \sum_{i=1}^n u_i \ln \frac{x_{t,i}}{\hat{y}_t} \\ &= \left(\sum_{i=1}^n u_i \ln x_{t,i} \right) - \ln(\hat{y}_t) = L_{\text{en}}(y_t, \hat{y}_t) - \sum_{i=1}^n u_i L_{\text{en}}(y_t, x_{t,i}) \end{aligned}$$

by symmetry we also have the case $y = 0$ demonstrating (1).

Weighted Average Algorithm - Proof continued

Proof – 2

Now observe that (1) is a telescoping equality and we have

$$\sum_{t=1}^m L_{\text{en}}(y_t, \hat{y}_t) - \sum_{t=1}^m \sum_{i=1}^n u_i L_{\text{en}}(y_t, x_{t,i}) = d(\mathbf{u}, \mathbf{v}_1) - d(\mathbf{u}, \mathbf{v}_{m+1})$$

Now since since the above holds for any $\mathbf{u} \in \Delta_n$ in particular the unit vectors $(1, 0, \dots, 0), (0, 1, \dots, 0), \dots, (0, 0, \dots, 1)$ and then if we upper bound by noting that $d(\mathbf{u}, \mathbf{v}_1) \leq \ln n$ and $-d(\mathbf{u}, \mathbf{v}_{m+1}) \leq 0$ we have proved theorem. □

Weighted Average Algorithm - HEDGE

HEDGE was introduced in [FS97], generalising the weighted majority analysis to the *allocation* setting.

Allocation setting

On each trial the learner **plays** an allocation $\mathbf{v}_t \in \Delta_n$, then nature returns a loss vector ℓ_t . I.e., the loss of expert i is $\ell_{t,i}$.

Two models for the learner's **play**:

1. We simply incur loss so that $L_A(t) := \mathbf{v}_t \cdot \ell_t$.
2. Alternately learner randomly selects an action $\hat{y} \in [n]$ according to the discrete distribution over $[n]$ so that $\text{Prob}(j) := v_{t,j}$. Thus

$$E[L_A(t)] = E[L_{t,\hat{y}}] = \mathbf{v}_t \cdot \ell_t.$$

- Observe that this setting can *simulate* the setting where we receive side-information \mathbf{x}_t and have a fixed loss function.
- For the randomised setting the “mechanism” generating the loss vectors ℓ_t must be oblivious to the learner's selection \hat{y} until trial $t + 1$.

HEDGE - Theorem

Theorem Hedge (Bound) [LW94,FS97]

For all sequence of loss vectors

$$S = \ell_1, \dots, \ell_m \in [0, 1]^n$$

the regret of the *weighted average* WA algorithm with $\eta = \sqrt{2 \ln n / m}$ is

$$E[L_{\text{WA}}(S)] - \min_i L_i(S) \leq \sqrt{2m \ln n}.$$

HEDGE Theorem - Proof (1)

Proof – 1

We first prove the following “progress versus regret” inequality.

$$\mathbf{v}_t \cdot \boldsymbol{\ell}_t - \mathbf{u} \cdot \boldsymbol{\ell}_t \leq \frac{1}{\eta} (d(\mathbf{u}, \mathbf{v}_t) - d(\mathbf{u}, \mathbf{v}_{t+1})) + \frac{\eta}{2} \sum_{i=1}^n v_{t,i} \ell_{t,i}^2 \text{ for all } \mathbf{u} \in \Delta_n. \quad (2)$$

Let $Z_t := \sum_{i=1}^n v_{t,i} \exp(-\eta \ell_{t,i})$. Observe that

$$\begin{aligned} d(\mathbf{u}, \mathbf{v}_t) - d(\mathbf{u}, \mathbf{v}_{t+1}) &= \sum_{i=1}^n u_i \ln \frac{v_{t+1,i}}{v_{t,i}} \\ &= -\eta \sum_{i=1}^n u_i \ell_{t,i} - \sum_{i=1}^n u_i \ln Z_t \\ &= -\eta \mathbf{u} \cdot \boldsymbol{\ell}_t - \ln \sum_{i=1}^n v_{t,i} \exp(-\eta \ell_{t,i}) \\ &\geq -\eta \mathbf{u} \cdot \boldsymbol{\ell}_t - \ln \sum_{i=1}^n v_{t,i} \left(1 - \eta \ell_{t,i} + \frac{1}{2} \eta^2 \ell_{t,i}^2\right) \end{aligned} \quad (3)$$

$$\begin{aligned} &= -\eta \mathbf{u} \cdot \boldsymbol{\ell}_t - \ln \left(1 - \eta \mathbf{v}_t \cdot \boldsymbol{\ell}_t + \frac{1}{2} \eta^2 \sum_{i=1}^n v_{t,i} \ell_{t,i}^2\right) \\ &\geq \eta (\mathbf{v}_t \cdot \boldsymbol{\ell}_t - \mathbf{u} \cdot \boldsymbol{\ell}_t) - \frac{1}{2} \eta^2 \sum_{i=1}^n v_{t,i} \ell_{t,i}^2 \end{aligned} \quad (4)$$

Using inequalities $e^{-x} \leq 1 - x + \frac{x^2}{2}$ for $x \geq 0$ and $\ln(1+x) \leq x$ for (3) and (4) demonstrating (2).

HEDGE Theorem - Proof (2)

Proof – 2

Summing over t and rearranging we have

$$\begin{aligned} \sum_{t=1}^m (\mathbf{v}_t \cdot \boldsymbol{\ell}_t - \mathbf{u} \cdot \boldsymbol{\ell}_t) &\leq \frac{1}{\eta} (d(\mathbf{u}, \mathbf{v}_1) - d(\mathbf{u}, \mathbf{v}_{m+1})) + \frac{\eta}{2} \sum_{t=1}^m \sum_{i=1}^n v_{t,i} \ell_{t,i}^2 \\ &\leq \frac{\ln n}{\eta} + \frac{\eta}{2} \sum_{t=1}^m \sum_{i=1}^n v_{t,i} \ell_{t,i}^2 \end{aligned} \tag{5}$$

Now since since the above holds for any $\mathbf{u} \in \Delta_n$ it then holds in particular for the unit vectors $(1, 0, \dots, 0), (0, 1, \dots, 0), \dots, (0, 0, \dots, 1)$ and then we upper bound by noting that $d(\mathbf{u}, \mathbf{v}_1) \leq \ln n$, $-d(\mathbf{u}, \mathbf{v}_{m+1}) \leq 0$, and

$\sum_{t=1}^m \sum_{i=1}^n v_{t,i} \ell_{t,i}^2 \leq m$. Finally we “tune” by choosing $\eta = \sqrt{2 \ln n / m}$ and we have proved theorem. \square

Question: how can we the above to prove a theorem if the loss is now in the range $[0, B]$?

Comments

- Easy to combine many pretty good experts (algorithms) so that Master is guaranteed to be almost as good as the best
- Bounds **logarithmic** in number of experts. Use many experts! Limits only in computational resources.
- Observe updating is **multiplicative**
- There exist algorithms [V90] with improved (smaller) constants for b with a more complex prediction strategies than weighted average.

Next: So far we have given bounds which grow slowly in the number of experts. The only significant drawback is potentially computational if we wish to work with large classes of experts. With this in mind we may wish to work with *structured* sets of experts for either a computational advantages or advantages in bound. We consider now **linear combinations** of experts (linear classifiers).

Part II

Online learning of linear classifiers

A more general setting (1)

Instance	Prediction of alg A	Label	Loss of alg A	
\mathbf{x}_1	\hat{y}_1	y_1	$L(y_1, \hat{y}_1)$	
\vdots	\vdots	\vdots	\vdots	
\mathbf{x}_t	\hat{y}_t	y_t	$L(y_t, \hat{y}_t)$	Sequence of examples
\vdots	\vdots	\vdots	\vdots	
\mathbf{x}_m	\hat{y}_m	y_m	$L(y_m, \hat{y}_m)$	
				<hr/>
Total Loss			$L_A(S)$	

$$S = (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$$

Comparison class $\mathcal{U} = \{\mathbf{u}\}$ (AKA hypothesis space, concept class)

Relative loss (Regret)

$$L_A(S) - \inf_{\{\mathbf{u} \in \mathcal{U}\}} \text{Loss}_{\mathbf{u}}(S)$$

Goal: Bound relative loss for arbitrary sequence S

A more general setting (2)

Now

- We consider the case where \mathcal{U} is a set of *linear threshold* function.
- For simplicity we will focus on the case where there exists a $\mathbf{u} \in \mathcal{U}$ s.t. $\text{Loss}_{\mathbf{u}}(S) = 0$. This is known as *realizable* case. Compare to the previously considered halving algorithm versus weighted majority algorithm.
- We will later see how a linear threshold function may be used to represent a *disjunction* or *conjunction*.

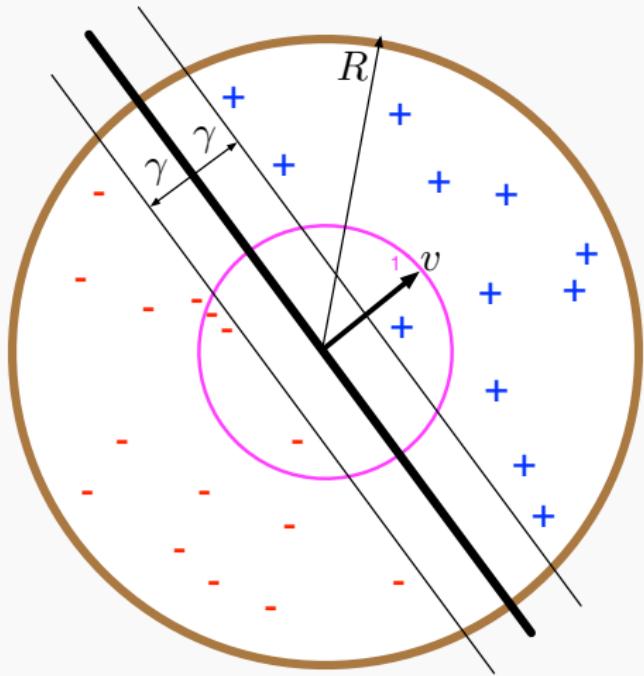
Preview : Solutions!

- We will now develop two efficient algorithms for learning disjunctions, more generally linearly threshold functions
- The algorithms will be efficient with only one weight per literal maintained.
- Each update will be $O(1)$ time per literal.
- Our solutions will be the PERCEPTRON and the WINNOW algorithm
- The WINNOW algorithm will have the advantage of a better performance guarantee at least on disjunctions
- The PERCEPTRON algorithm will have the advantage of compatibility with the “kernel trick”

Perceptron

The Perceptron set-up

Assumption: Data is linearly separable by some margin γ . Hence exists a hyperplane with normal vector \mathbf{v} such that



1. $\|\mathbf{v}\| = 1$
2. All examples (\mathbf{x}_t, y_t)
 - $\forall y_t \quad y_t \in \{-1, +1\}$.
 - $\forall \mathbf{x}_t, \quad \|\mathbf{x}_t\| \leq R$.
3. $\forall (\mathbf{x}_t, y_t), y_t(\mathbf{x}_t \cdot \mathbf{v}) \geq \gamma$

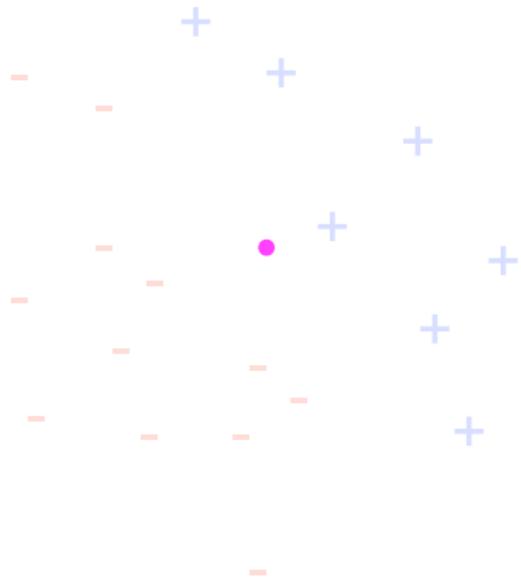
The PERCEPTRON learning algorithm

PERCEPTRON ALGORITHM

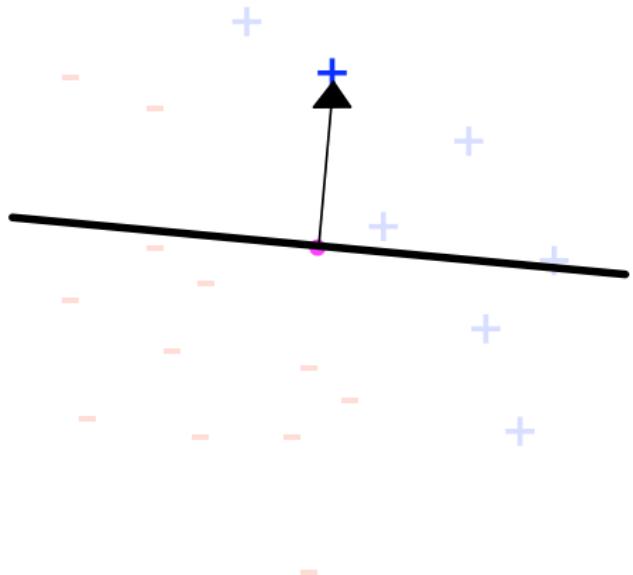
Input: $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m) \in \mathbf{R}^n \times \{-1, 1\}$

1. Initialise $\mathbf{w}_1 = \vec{0}; M_1 = 0$.
2. For $t = 1$ to m do
3. Receive pattern: $\mathbf{x}_t \in \mathbf{R}^n$
4. Predict: $\hat{y}_t = \text{sign}(\mathbf{w}_t \cdot \mathbf{x}_t)$
5. Receive label: y_t
6. If mistake ($\hat{y}_t y_t \leq 0$)
 - Then Update $\mathbf{w}_{t+1} = \mathbf{w}_t + y_t \mathbf{x}_t; M_{t+1} = M_t + 1$
 - Else $\mathbf{w}_{t+1} = \mathbf{w}_t; M_{t+1} = M_t$.
7. End For

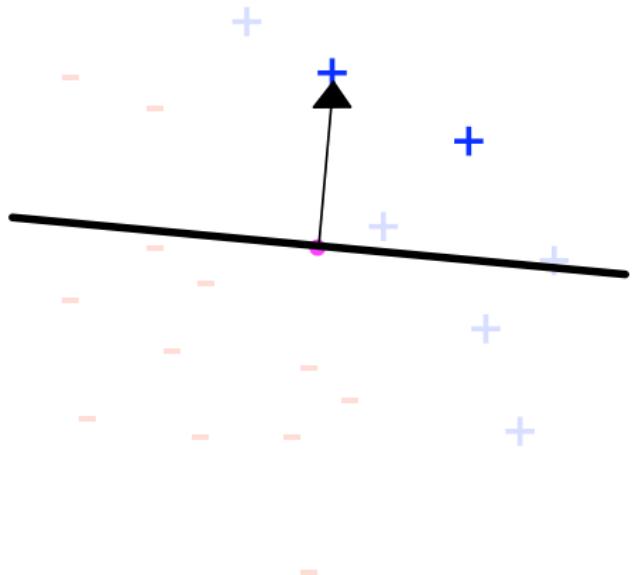
Example: trace for the PERCEPTRON algorithm



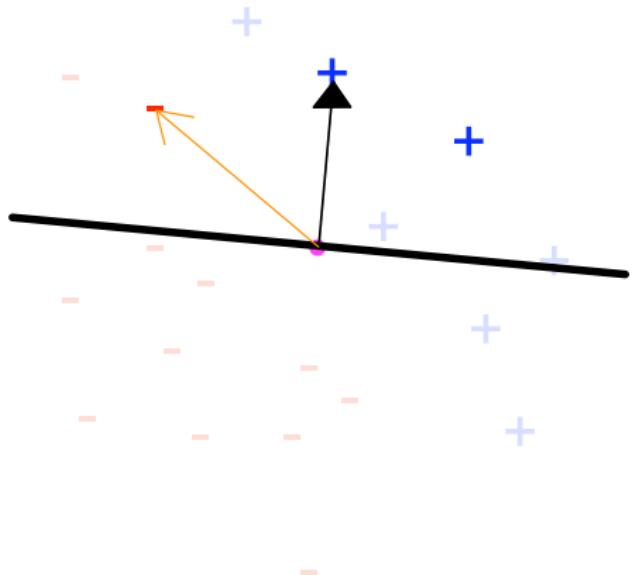
Example: trace for the PERCEPTRON algorithm



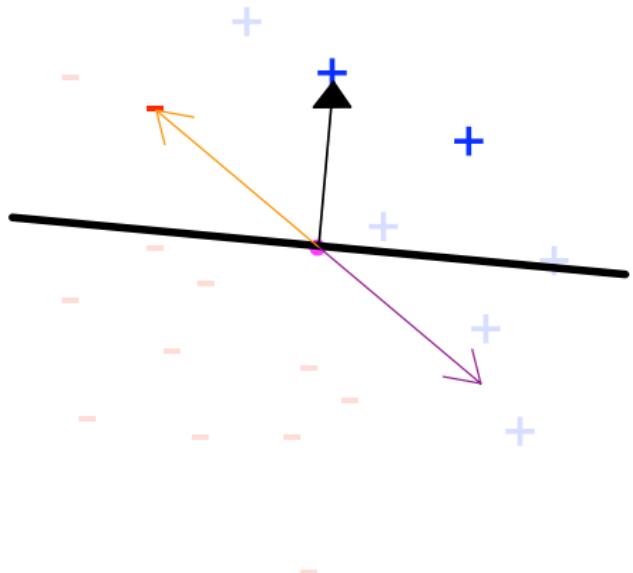
Example: trace for the PERCEPTRON algorithm



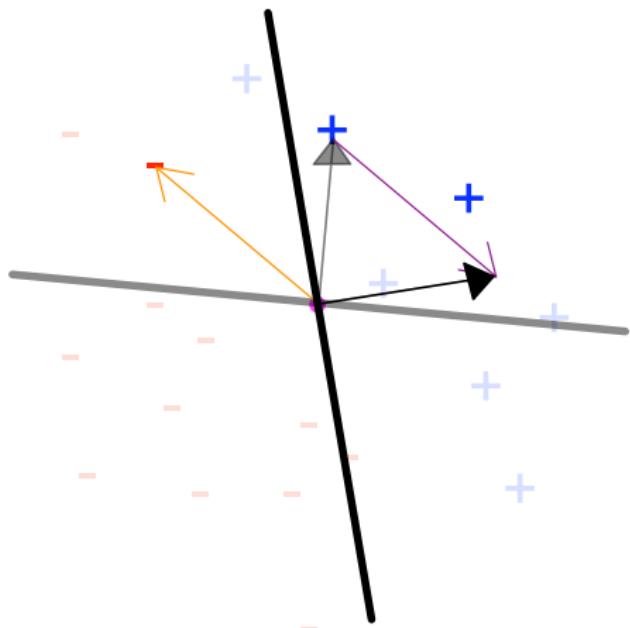
Example: trace for the PERCEPTRON algorithm



Example: trace for the PERCEPTRON algorithm



Example: trace for the PERCEPTRON algorithm



Bound on number of mistakes

- The number of mistakes that the perceptron algorithm can make is at most $\left(\frac{R}{\gamma}\right)^2$.
- Proof by combining upper and lower bounds on $\|\mathbf{w}\|$.

Pythagorean Lemma

On trials where “mistakes” occur we have the following inequality,

Lemma: If $(\mathbf{w}_t \cdot \mathbf{x}_t)y_t < 0$ then $\|\mathbf{w}_{t+1}\|^2 \leq \|\mathbf{w}_t\|^2 + \|\mathbf{x}_t\|^2$

Proof:

$$\begin{aligned}\|\mathbf{w}_{t+1}\|^2 &= \|\mathbf{w}_t + y_t \mathbf{x}_t\|^2 \\ &= \|\mathbf{w}_t\|^2 + 2(\mathbf{w}_t \cdot \mathbf{x}_t)y_t + \|\mathbf{x}_t\|^2 \\ &\leq \|\mathbf{w}_t\|^2 + \|\mathbf{x}_t\|^2\end{aligned}$$

Upper bound on $\|\mathbf{w}_t\|$

Lemma: $\|\mathbf{w}_t\|^2 \leq M_t R^2$

Proof: By induction

- Claim: $\|\mathbf{w}_t\|^2 \leq M_t R^2$
- Base: $M_1 = 0$, $\|\mathbf{w}_1\|^2 = 0$
- Induction step (assume for t and prove for $t + 1$) when we have a mistake on trial t :

$$\|\mathbf{w}_{t+1}\|^2 \leq \|\mathbf{w}_t\|^2 + \|\mathbf{x}_t\|^2 \leq \|\mathbf{w}_t\|^2 + R^2 \leq (M_{t+1})R^2$$

Here we used the Pythagorean lemma. If there is no mistake, then trivially $\mathbf{w}_{t+1} = \mathbf{w}_t$ and $M_{t+1} = M_t$.

Lower bound on $\|\mathbf{w}_t\|$

Lemma: $M_t \gamma \leq \|\mathbf{w}_t\|$

Observe: $\|\mathbf{w}_t\| \geq \mathbf{w}_t \cdot \mathbf{v}$ because $\|\mathbf{v}\| = 1$. (Cauchy-Schwarz)

We prove a lower bound on $\mathbf{w}_t \cdot \mathbf{v}$ using induction over t

- Claim: $\mathbf{w}_t \cdot \mathbf{v} \geq M_t \gamma$
- Base: $t = 1$, $\mathbf{w}_1 \cdot \mathbf{v} = 0$
- Induction step (assume for t and prove for $t + 1$):
If mistake then

$$\begin{aligned}\mathbf{w}_{t+1} \cdot \mathbf{v} &= (\mathbf{w}_t + \mathbf{x}_t y_t) \cdot \mathbf{v} \\ &= \mathbf{w}_t \cdot \mathbf{v} + y_t \mathbf{x}_t \cdot \mathbf{v} \\ &\geq M_t \gamma + \gamma \\ &= (M_t + 1)\gamma\end{aligned}$$

Combining the upper and lower bounds

Let $M := M_{m+1}$ denote the total number of updates (“mistakes”) then

$$(M\gamma)^2 \leq \|\mathbf{w}_{m+1}\|^2 \leq MR^2$$

Thus simplifying we have the famous ...

Theorem (Perceptron Bound [Novikoff])

For all sequences of examples

$$S = (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m) \in \mathbf{R}^n \times \{-1, 1\}$$

the mistakes of the PERCEPTRON algorithm is bounded by

$$M \leq \left(\frac{R}{\gamma}\right)^2,$$

with $R := \max_t \|\mathbf{x}_t\|$. If there exists a vector \mathbf{v} with $\|\mathbf{v}\| = 1$ and constant γ such that $(\mathbf{v} \cdot \mathbf{x}_t)y_t \geq \gamma$.

Comments

Comments

- It is often convenient to express the bound in the following form.

Here define $\mathbf{u} := \frac{\mathbf{v}}{\gamma}$ then

$$M \leq R^2 \|\mathbf{u}\|^2 \quad (\forall \mathbf{u} : (\mathbf{u} \cdot \mathbf{x}_t) y_t \geq 1)$$

- Suppose we have *linearly separable* data set S . Questions:
 - Observe that \mathbf{w}_{m+1} does not necessarily linearly separate S . Why?
 - How can we use the PERCEPTRON to find a vector \mathbf{w} that separates S ?
 - How long will this computation take?
- There are variants on the PERCEPTRON that operate on a single example at a time that converge to the “SVM” max-margin linear separator.

Interlude: Regret Bounds for Linear Separation

Going Deeper : Regret Bounds for Linear Separation

Recall the regularisation approach to supervised learning.

$$h^* = \arg \min_{h \in \mathcal{H}} \sum_{t=1}^m L(y_t, h(\mathbf{x}_t)) + \lambda \text{penalty}(h)$$

Example: Ridge Regression

$$\arg \min_{\mathbf{w} \in \mathbb{R}^n} \sum_{t=1}^m L(y_t, \mathbf{w} \cdot \mathbf{x}_t) + \lambda \|\mathbf{w}\|^2$$

Example: Soft Margin SVM

$$\arg \min_{\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}} \sum_{t=1}^m L_{\text{hi}}(y_t, \mathbf{w} \cdot \mathbf{x}_t + b) + \lambda \|\mathbf{w}\|^2$$

with $L_{\text{hi}}(y, \hat{y}) := \max(0, 1 - y\hat{y})$.

Online Approach

Recall the regularisation approach to supervised learning.

BATCH :

$$\arg \min_{h \in \mathcal{H}} \sum_{t=1}^m L(y_t, h(\mathbf{x}_t)) + \lambda \text{penalty}(h)$$

ONLINE commonly motivated via

$$h_{t+1} = \arg \min_{h \in \mathcal{H}} L(y_t, h(\mathbf{x}_t)) + \lambda \text{penalty}(h, h_t)$$

Motivation?

Online Gradient Descent with Hinge Loss and $\|\cdot\|_2^2$ penalty

$$\mathbf{w}_{t+1} = \arg \min_{\mathbf{w} \in \mathbb{R}^n} L_{\text{hi}}(y_t, \mathbf{w} \cdot \mathbf{x}_t) + \lambda \|\mathbf{w} - \mathbf{w}_t\|^2$$

Solving for the update (“take derivative and set to zero”). Recalling
 $L_{\text{hi}}(y, \hat{y}) := \max(0, 1 - y\hat{y})$.

$$\mathbf{w}_{t+1} = \begin{cases} \mathbf{w}_t & y_t(\mathbf{w} \cdot \mathbf{x}_t) > 1 \\ \mathbf{w}_t + \frac{y_t \mathbf{x}_t}{2\lambda} & y_t(\mathbf{w} \cdot \mathbf{x}_t) < 1 \end{cases}$$

OGD with Hinge Loss and $\|\cdot\|_2^2$ penalty

OGD Algorithm

Initialise : $\mathbf{w}_1 := \mathbf{0}$, $L_{\text{OGD}} := 0$

Select: $\eta \in (0, \infty)$

For $t = 1$ To m Do

 Receive instance $\mathbf{x}_t \in \mathbb{R}^n$

 Predict $\hat{y}_t := \mathbf{w}_t \cdot \mathbf{x}_t$

 Receive label $y_t \in \{-1, 1\}$

 Incur loss $L_{\text{OGD}} := L_{\text{OGD}} + L_{\text{hi}}(y_t, \hat{y}_t)$

 Update weights $\mathbf{w}_{t+1} := \mathbf{w}_t + [y_t \hat{y}_t < 1] \eta y_t \mathbf{x}_t$.

How does the above differ from the perceptron?

Regret Bound for OGD

Theorem (Based on [G03])

Given $R = \max_t \|\mathbf{x}_t\|$, $\|\mathbf{u}\| \leq U$ and $\eta := \frac{U}{R\sqrt{m}}$ we have that for Algorithm OGD,

$$\sum_{t=1}^m L_{\text{hi}}(y_t, \hat{y}_t) - L_{\text{hi}}(y_t, \mathbf{u} \cdot \mathbf{x}_t) \leq \sqrt{U^2 R^2 m}, \quad (6)$$

for any vector \mathbf{u} .

Regret Bound for OGD – Proof (1)

Proof

Using the convexity of the hinge loss (wrt its 2nd argument), we have

$$L_{\text{hi}}(y_t, \hat{y}_t) - L_{\text{hi}}(y_t, \mathbf{u} \cdot \mathbf{x}_t) \leq (\mathbf{w}_t - \mathbf{u}) \cdot \mathbf{z}_t, \quad (7)$$

where

$$\mathbf{z}_t := -y_t \mathbf{x}_t [y_t(\mathbf{w}_t \cdot \mathbf{x}_t) < 1] \in \nabla_{\mathbf{w}} L_{\text{hi}}(y_t, \mathbf{w} \cdot \mathbf{x}_t).$$

From the update we have,

$$\|\mathbf{w}_{t+1} - \mathbf{u}\|^2 = \|\mathbf{w}_t - \eta \mathbf{z}_t - \mathbf{u}\|^2 = \|\mathbf{w}_t - \mathbf{u}\|^2 - 2\eta \langle \mathbf{w}_t - \mathbf{u}, \mathbf{z}_t \rangle + \eta^2 \|\mathbf{z}_t\|^2.$$

Thus

$$\langle \mathbf{w}_t - \mathbf{u}, \mathbf{z}_t \rangle = \frac{1}{2\eta} \left(\|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}_{t+1} - \mathbf{u}\|^2 + \eta^2 \|\mathbf{z}_t\|^2 \right). \quad (8)$$

Regret Bound for OGD – Proof (2)

Proof – Continued

From (8) we have

$$\begin{aligned}\sum_{t=1}^m \langle \mathbf{w}_t - \mathbf{u}, \mathbf{z}_t \rangle &= \sum_{t=1}^m \frac{1}{2\eta} \left(\|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}_{t+1} - \mathbf{u}\|^2 + \eta^2 \|\mathbf{z}_t\|^2 \right) \\ &\leq \frac{1}{2\eta} \left(\|\mathbf{u}\|^2 + \eta^2 \sum_{t=1}^m \|\mathbf{z}_t\|^2 \right) \\ &= \frac{1}{2\eta} \|\mathbf{u}\|^2 + \frac{\eta}{2} \sum_{t=1}^m \|\mathbf{x}_t\|^2 [y_t(\mathbf{w}_t \cdot \mathbf{x}_t) < 1] \\ &\leq \frac{1}{2\eta} U^2 + \frac{\eta}{2} m R^2 \\ &= \sqrt{U^2 R^2 m} \quad (\text{recall } \eta := \frac{U}{R\sqrt{m}})\end{aligned}$$

Lower bounding the L.H.S. with (7) and we are done. ■

Compare to the proof of HEDGE.

Deriving the perceptron algorithm/bound from OGD

The perceptron bound can be arrived by an analysis of OGD.

1. Observe that equation (6) implies,

$$\sum_{t=1}^m [y_t \neq \text{sign}(\hat{y}_t)] - L_{\text{hi}}(y_t, \mathbf{u} \cdot \mathbf{x}_t) \leq \sqrt{U^2 R^2 m}.$$

2. Now assume there exists a linear classifier \mathbf{u} such that $y_t(\mathbf{u} \cdot \mathbf{x}_t) \geq 1$ for all $t = 1, \dots, m$. Thus,

$$\sum_{t=1}^m [y_t \neq \text{sign}(\hat{y}_t)] \leq \sqrt{U^2 R^2 m}.$$

3. Now make OGD **conservative** that is we only update when $y_t \hat{y}_t \leq 0$ versus $y_t \hat{y}_t \leq 1$ i.e., trials in which a mistake is made.
4. Thus with respect to the bound we can ignore the trials where a mistake is not made so that we can set $m = M := \sum_{t=1}^m [y_t \neq \text{sign}(\hat{y}_t)]$ which implies

$$M \leq \sqrt{U^2 R^2 M} \longrightarrow M \leq U^2 R^2$$

5. Finally note that the sign of the predictions is unchanged (**why?**) for any value of $\eta > 0$ thus we may set $\eta = 1$.

Comparing Perceptron and Winnow via Disjunction

Learning

Boolean functions

Claim

A focus of classical AI is on the representation, learning and manipulation of **symbolic** knowledge. Perhaps the simplest symbolic knowledge representation scheme is *boolean logic*.

- A boolean function f may be represented as a map
$$f : \{\text{true}, \text{false}\}^n \rightarrow \{\text{true}, \text{false}\}.$$
- In what follows we will always associate true with the number 1 however we use either -1 or 0 to represent false as numerically convenient.

Base boolean functions

(here $\text{true} = 1$ and $\text{false} = 0$)

1. $x_1 \wedge x_2 := x_1 x_2$ ("and")
2. $x_1 \vee x_2 := \text{sign}(x_1 + x_2)$ ("or")
3. $\bar{x}_1 := 1 - x_1$ ("not")

Comparison classes of boolean functions (1)

Terminology:

1. A single (negated) variable is known as a *literal* (e.g., x_1, x_3, x_7, \bar{x}_5)
2. A *term* or *conjunction* of literals is an iterated “and” applied to the literals (e.g., $x_1x_3, \bar{x}_4x_5x_7$)
3. A *clause* or *disjunction* of literals is an iterated “or” applied to literals (e.g., $x_1 \vee x_3, \bar{x}_4 \vee x_5 \vee x_7$)
4. *Monotone* disjunction or conjunction implies no negated literals

Questions : Given n variables denote the comparison class of all possible terms as \mathcal{U}_\wedge and clauses as \mathcal{U}_\vee . What are their cardinalities i.e. $|\mathcal{U}_\wedge|$ and $|\mathcal{U}_\vee|$? What is the cardinality of the set of all n variable boolean functions?

Comparison classes of boolean functions (2)

Recall that a linear threshold $f_{\mathbf{u}, b} : \mathbf{R}^n \rightarrow \{-1, 1\}$ function may be written as,

$$f_{\mathbf{u}, b}(\mathbf{x}) := \text{sign}(\mathbf{u} \cdot \mathbf{x} + b),$$

i.e. those functions determined by a separating hyperplane. The comparison class of all linear threshold functions is

$$\mathcal{U}_{\text{lt}} := \{f_{\mathbf{u}, b} : \mathbf{u} \in \mathbf{R}^n, b \in \mathbf{R}\}$$

Question: How can we use comparison class of linear-thresholded functions after the feature map to represent monotone disjunctions? and monotone conjunctions?

Feature map: We can use the feature map

$$\phi(\mathbf{x}) := (x_1, 1 - x_1, \dots, x_n, 1 - x_n)$$

in order to represent *non-monotone* disjunctions and conjunctions. (**How does this work?**)

Example : Representing a monotone disjunction as a linear threshold function

variables/experts				<i>true label</i>	$x_1 \vee x_3$	$x_3 \vee x_4$
$x_{t,1}$	$x_{t,2}$	$x_{t,3}$	$x_{t,4}$			
1	1	0	0	0	1	0
1	0	1	0	1	1	1
0	1	1	1	0	1	1
0	1	0	0	1	0	0
					↑	↑
					3	2

mistakes

$x_1 \vee x_3$ becomes $\mathbf{u} = (1, 0, 1, 0)$ and $b = -1/2$ i.e.,

$$[\text{sign}(\mathbf{u} \cdot \mathbf{x}_t - 1/2) == 1] = x_{t,1} \vee x_{t,3}$$

A suboptimal solution

Problem

Goal: An algorithm to predict as well as best k -literal (monotone) disjunction (over n variables).

Solution

Use weighted majority where each disjunction is an expert maintain one weight per disjunction: thus $\binom{n}{k}$ weights.

$$\text{Mistakes of WM} \leq 2.63 M + 2.63 k \ln \frac{ne}{k}$$

Where M is number of mistakes of best disjunction.

Note

- We used the inequality $\binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$
- Time (and space) exponential in k
- Our goal: efficient algs: one weight per literal

Perceptron algorithm for disjunctions – 1

Corollary

With the feature map $\phi(\mathbf{x}) := (\mathbf{x}, 1)$ we may use the perceptron to learn monotone disjunctions so that

$$M \leq (4k + 1)(n + 1)$$

Where k is the number of literals out the n possible literals.

Note: There exists a generic lower bound for rotation invariant alg
(perceptron and svm are examples):

$$M = \Omega(n)$$

See : *The Perceptron algorithm vs. Winnow: linear vs. logarithmic mistake bounds when few input variables are relevant* (Kivinen and Warmuth, 1995).

Perceptron algorithm for disjunctions – 2

Proof

We will use the following form of the perceptron bound,

$$M \leq R^2 \|\mathbf{u}\|^2 \quad (\forall \mathbf{u} : (\mathbf{u} \cdot \mathbf{x}_t) y_t \geq 1)$$

Assume $\mathbf{x} \in \{0, 1\}^n$ then with the feature map $\phi(\mathbf{x}) := (\mathbf{x}, 1)$. We claim the following $\mathbf{u}^* \in \mathbb{R}^{n+1}$ separates with a margin of 1,

i.e.

$$u_i^* := \begin{cases} 2 & i \text{ is a literal} \\ 0 & i \text{ is not a literal} \\ -1 & i \text{ is the bias weight} \end{cases} .$$

1. $\mathbf{u}^* \cdot \phi(\mathbf{x}) \geq 1$ (positive examples, $y_t = 1$)
2. $\mathbf{u}^* \cdot \phi(\mathbf{x}) = -1$ (negative examples, $y_t = -1$)

Note that since $\mathbf{x} \in \{0, 1\}^n$ then $\|\phi(\mathbf{x})\|^2 \leq (n + 1)$ and $\|\mathbf{u}^*\|^2 = 4k + 1$. Thus $M \leq (4k + 1)(n + 1)$. □

WINNOW algorithm

Input: $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m) \in \{0, 1\}^n \times \{0, 1\}$

1. Initialise $\mathbf{w}_1 = \vec{1}$.
2. For $t = 1$ to m do
3. Receive pattern: $\mathbf{x}_t \in \{0, 1\}^n$
4. Predict:

$$\hat{y}_t = \begin{cases} 0 & \mathbf{w}_t \cdot \mathbf{x}_t < n \\ 1 & \mathbf{w}_t \cdot \mathbf{x}_t \geq n \end{cases}$$

5. Receive label: $y_t \in \{0, 1\}$
6. If mistake ($\hat{y}_t \neq y_t$)
 - Update: $\mathbf{w}_{t+1,i} = \mathbf{w}_{t,i} 2^{(y_t - \hat{y}_t) x_{t,i}}$ $(1 \leq i \leq n)$.
7. End For

Mistake bound of WINNOW

Theorem: Mistake bound of WINNOW (Littlestone)

The mistakes of WINNOW may be bounded by

$$M \leq 3k(\log n + 1) + 2$$

If there exists a *consistent k-literal monotone disjunction*.

Observation

There is an exponential improvement in bound (over the perceptron) with respect to the dimension n in the upper bound for disjunction learning. Although we will not give the bound for linear threshold learning with WINNOW in that case the bounds are incomparable (wrt PERCEPTRON) where WINNOW (like LASSO) prefers *sparse* hypotheses.

Proof of WINNOW Bound

Bound on “mistakes” on positive examples. ($y_t = 1$)

1. On a mistake : at least one *relevant* weight is doubled.
2. Relevant weights never decrease.
3. Once a relevant weight $w_{t,i} \geq n$ it will no longer change

Conclusion: Mistakes on positive examples $M_p \leq k(\log n + 1)$

Bound on “mistakes” on negative examples. ($y_t = 0$)

Let $W_t = \sum_{i=1}^n w_{t,i}$. Denote M_f as mistakes on negative examples.

1. $W_1 = n$
2. On a positive mistake ($y_t = 1$) $W_{t+1} \leq W_t + n$
3. On a negative mistake ($y_t = 0$) $W_{t+1} \leq W_t - \frac{n}{2}$
4. Combining $W_{m+1} \leq n + M_p n - M_f \frac{n}{2}$
5. Thus $M_f \leq 2k(\log n + 1) + 2$.

Theorem: $M \leq M_p + M_f \leq 2 + 3k(\log n + 1)$

Note: By *relevant* we mean a weight that corresponds to a literal in the disjunction.

Online bounds give Batch Bounds

- We can use online bounds to give bounds in a batch setting.
- Suppose the data is drawn from a distribution P and a mistake bound for algorithm \mathcal{A} for any such set is B .

Theorem

Given m , then let t be drawn uniformly at random from $\{1, \dots, m\}$. Let \mathcal{S} consist of t examples sampled iid from P , let (\mathbf{x}', y') be an additional example sampled from P then

$$\text{Prob}(\mathcal{A}_{\mathcal{S}}(\mathbf{x}') \neq y') \leq \frac{B}{m}$$

with respect to the draw of t , \mathcal{S} , and (\mathbf{x}', y') .

Proof

There are no more than B “trials” with mistakes therefore since t is drawn uniformly from $\{1, \dots, m\}$ there is no more than a B/m probability of hitting a trial with a mistake.

Note: More sophisticated online to batch conversions exist with stronger properties.

Case study : DNF with PERCEPTRON and WINNOW

In the following we will ...

1. We note that *surprisingly* the previous mistake bounds of PERCEPTRON and WINNOW are stated in terms of the minimally consistent disjunction. However finding such disjunctions is NP-complete.
2. Then we will compare the time complexity and mistake bounds for learning boolean DNF functions which are a natural and complete class of boolean functions. (Whether DNF is PAC Learnable under the uniform distribution has been an open problem for 25+ years!!)

Finding maximally sparse classifiers is hard (1)

Sparsity is important concept in machine learning. Here the idea is to find a classifier which depends on the minimal number of features. Such a classifier has the two-fold advantage.

1. In general it will generalize well to new data
2. It may be useful to discover which features are particularly predictive

Finding maximally sparse classifiers is hard (2)

A common method to encourage sparsity (L1 Objective) is to minimise

$$\arg \min_{\mathbf{w}} \sum_{t=1}^m [1 - y_t(\mathbf{x}_t \cdot \mathbf{w})]_+ + \lambda \|\mathbf{w}\|_1$$

where this is a proxy for the nonconvex objective

$$\arg \min_{\mathbf{w}} \sum_{t=1}^m [1 - y_t(\mathbf{x}_t \cdot \mathbf{w})]_+ + \lambda \|\mathbf{w}\|_p \text{ as } p \rightarrow 0$$

We will argue the above is hard in the limit $\lambda \rightarrow 0$.

Note $[a]_+ := a\mathcal{I}[a > 0]$.

Observation (finding maximally sparse linear classifiers):

A simple instance of feature selection would be to find the minimal k -literal disjunction consistent with a data set. However we will see that the decision problem is **NP-complete**.

Finding maximally sparse classifiers is hard (3)

Decision problem (k -consistent disjunction): Given examples $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m) \in \{0, 1\}^n \times \{0, 1\}$ does there exist a consistent k -literal disjunction? i.e., does there exist $\mathbf{u} \in \{0, 1\}^n$ s.t.

$$(\mathbf{u} \cdot \mathbf{x} \geq 1 \iff y_t = 1) \text{ and } (\mathbf{u} \cdot \mathbf{x} = 0 \iff y_t = 0)$$

and $\|\mathbf{u}\|^2 \leq k$.

Theorem: consistent k -disjunction is NP-complete

The proof works by the following sequence of reductions.

$$\text{CNFSAT} \leq_p k\text{-set cover} \leq_p k\text{-consistent disjunction}$$

Cook – Levin Theorem

CNF (Conjunctive Normal Form): is a conjunction of clauses. For example

$$(x_1 \vee x_4 \vee x_7) \wedge x_1 \wedge (\bar{x}_2 \vee x_2 \vee x_5)$$

CNFSAT – decision problem

Given a boolean function in conjunctive normal form does there exist an assignment to the variables so that the function is true.

Theorem

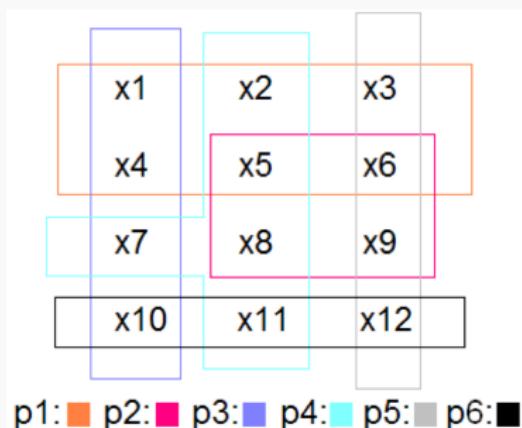
CNFSAT is **NP**-complete.

Observation: if $A \in \mathbf{NP}$ and $\text{CNFSAT} \leq_p A$ then A is **NP**-complete.

NP-complete (set cover) – 1

Theorem: k -set-cover is NP-complete

Given a set of m elements $X := \{x_1, x_2, \dots, x_m\}$, a collection of n subsets of elements $S_1, \dots, S_n \subseteq X$ and a positive integer k the problem of deciding if there exist k subsets from $S_{i_1}, \dots, S_{i_k} \in \{S_1, \dots, S_n\}$ that cover, $\bigcup_{j=1}^k S_{i_j} = X$, the m elements is NP-complete.



Set cover illustration from: "Lecture Comp 260: Advanced Algorithms,
Lenore Cowen Tufts University, Spring 2011"

NP-complete (set cover) – 2

Proof.

1. set cover $\in \text{NP}$. The verifier simply checks if the proposed cover is a cover.
2. We proceed to prove that CNFSAT \leq_p set cover. Any instance of CNFSAT may be written as follows. Let

$$L = \{x_1, \dots, x_m, \bar{x}_1, \dots, \bar{x}_m\}$$

be a set of literals and then

$$\{C_1, \dots, C_n\} \subseteq L$$

are n subsets of literals then a CNF formula may be written as,

$$\bigwedge_{i=1}^n (\bigvee_{x \in C_i} x)$$

NP-complete (set cover) – 3

Proof – continued.

From the CNF we construct a set cover instance as follows. We will construct a collection of sets $\mathcal{S} := \{S_1, \dots, S_m, \bar{S}_1, \dots, \bar{S}_m\}$ from the set of $n + m$ elements $E = \{c_1, \dots, c_n, z_1, \dots, z_m\}$ from the CNF formula $\bigwedge_{i=1}^n (\bigvee_{x \in C_i} x)$. Construct

$$S_i := \{c_j : x_i \in C_j\} \cup \{z_i\} \quad (1 \leq i \leq m)$$

$$\bar{S}_i := \{c_j : \bar{x}_i \in C_j\} \cup \{z_i\} \quad (1 \leq i \leq m)$$

and thus the setcover problem is

Does there exist m sets in \mathcal{S} that cover E ?

We now argue that for the proposed reduction that

yes-instance of k -set cover \iff yes-instance of CNFSAT

NP-complete (set cover) – 4

Proof – continued.

(set cover \Leftarrow CNFSAT) Observe that every **satisfying** assignment to the variables x_1, \dots, x_m corresponds to a m element set cover of S in particular if $\mathbf{x}^* \in \{t, f\}^m$ is a satisfying assignment then

$$\bigcup_{i=1}^m \{S_i : x_i^* = t\} \cup \{\bar{S}_i : x_i^* = f\}$$

is a cover of size m of E (**why?**).

(set cover \implies CNFSAT) Observe that every cover \mathcal{C} of size m corresponds to satisfying truth assignment, as we may construct the truth assignment,

$$x_i^* := \begin{cases} t & S_i \in \mathcal{C} \\ f & \bar{S}_i \in \mathcal{C} \end{cases} \quad (1 \leq i \leq m)$$

Note that S_i or \bar{S}_i is in \mathcal{C} otherwise element z_i is not covered; likewise both S_i or \bar{S}_i cannot be in \mathcal{C} because if they were some element z_j would not be covered (**why?**). The assignment \mathbf{x}^* must satisfy since every c_j is covered. \square

Set Cover \leq_p consistent disjunction

Proof sketch. The problem is of finding k -set cover reduces to the problem of finding a k -literal disjunction consistent with a set of examples. Let X be an $m \times n$ matrix of $m, n - \text{dimensional}$ “positive” examples then the n “columns” are the “sets” and a consistent disjunction is a set cover. As a data matrix (these are ‘positive’ examples thus $y_1 = \dots = y_{12} = 1$)

	P_1	P_2	P_3	P_4	P_5	P_6
x_1	1	0	1	0	0	0
x_2	1	0	0	1	0	0
x_3	1	0	0	0	1	0
x_4	1	0	1	1	0	0
x_5	1	1	0	1	0	0
x_6	1	1	0	0	1	0
x_7	0	0	1	1	0	0
x_8	0	1	0	1	0	0
x_9	0	1	0	0	1	0
x_{10}	0	0	1	0	0	1
x_{11}	0	0	0	1	0	1
x_{12}	0	0	0	0	1	1

Observe: $P_1 \vee P_2 \vee P_4 \vee P_6$ covers while $P_3 \vee P_4 \vee P_5$ is minimal.

Interestingly. The prediction bounds of winnow and the perceptron are in terms of the minimum consistent disjunction but neither find/learn or a minimal consistent disjunction.

Learning DNF with Winnow and the Perceptron – 1

DNF (Disjunctive Normal Form) corresponds to a simple boolean network with a single layer as such it may be learned by a neural network with a single hidden layer.

In the following we show that these boolean functions may be learned efficiently with the perceptron but with poor prediction performance. And inefficiently by winnow but with good prediction performance.

Learning DNF with Winnow and the Perceptron – 2

DNF (Disjunctive Normal Form): is a disjunction of terms. For example

$$x_1 x_4 x_7 \vee x_1 \bar{x}_2 \vee x_2 x_5$$

- DNF is then the set of all disjunctions of terms.
- DNF is a natural form for knowledge representation for example: the concept "cat" (from Lisa Hellerstein)

$$\begin{aligned} & (\text{IsHousePet} \wedge \text{Purrs} \wedge \text{HasTail}) \vee \\ & (\overline{\text{HasTail}} \wedge \text{Furry} \wedge \text{TaperedEars} \wedge \text{RoundEyes}) \vee \\ & (\text{NamedSylvester} \wedge \text{ChasesTweetyBird}) \end{aligned}$$

- Note all boolean functions may be represented as a DNF (Why?)
- Many unsolved problem in machine learning regarding DNF learning

k -term (monotone) DNF via Feature Expansion

$$\begin{array}{c} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_1 \\ x_2 \\ \vdots \\ x_n \\ \mathbf{x} = \begin{matrix} \vdots & \Rightarrow & \Phi(\mathbf{x}) = & x_1 x_2 \\ \vdots & & & \vdots \\ x_n & & & \vdots \\ & & & x_1 x_2 \dots x_n \end{matrix} \end{array}$$

n inputs

2^n features

k -term DNF in input space is k -literal disjunction in feature space

$$\underbrace{\Phi(\mathbf{x})}_{2^n} \cdot \underbrace{\Phi(\mathbf{y})}_{2^n} = \underbrace{\prod_{i=1}^n (1 + x_i y_i)}_{O(n) \text{ time}} = K_{\text{anova}}(\underbrace{\mathbf{x}}_n, \underbrace{\mathbf{y}}_n) \quad (\text{Simple ANOVA kernel})$$

Winnow versus Perceptron for k -term DNF

Perceptron: $w_t = \sum_{q \in \text{mistakes}} \alpha_q \Phi(\mathbf{x}_q)$

Prediction:

$$\begin{aligned} w_t \cdot \Phi(\mathbf{x}_t) &= \left(\sum_{q \in \text{mistakes}} \alpha_q \Phi(\mathbf{x}_q) \right) \cdot \Phi(\mathbf{x}_t) = \sum_{q \in \text{mistakes}} \alpha_q \Phi(\mathbf{x}_q) \cdot \Phi(\mathbf{x}_t) \\ &= \underbrace{\sum_{q \in \text{mistakes}} \alpha_q K(\mathbf{x}_q, \mathbf{x}_t)}_{\text{Prediction time: } O(n \cdot \# \text{ mistakes}) \leq O(nm)} \end{aligned}$$

Prediction time: $O(n \cdot \# \text{ mistakes}) \leq O(nm)$

Mistake bound: $O(k 2^n)$ (Why?)

Winnow: $w_{t,i} = \exp\left(-\eta \sum_{q \in \text{mistakes}} \alpha_q \Phi(\mathbf{x}_q)_i\right)$

log of weights is linear combination of past examples

Mistake bound: $O(k \ln 2^n) = O(k n)$ Prediction time: $\Omega(2^n \# \text{ mistakes})$

No kernel trick with purely mult. updates!, i.e., no obvious fast way to compute $w_t \cdot \Phi(\mathbf{x}_t)$ for WINNOW.

Summary: PERCEPTRON: (fast!, poor bound)

WINNOW: (slow, good bound!)

Summary so far

- Learning relative to best expert and best disjunction
- Learning relative to *linear combinations* of experts
- Linear combinations may represent Boolean functions
- PERCEPTRON versus WINNOW – similar algorithms very different performance when we consider a feature space expansion as applied to DNF
- We focused on boolean function learning for PERCEPTRON and WINNOW, however, they both learn linear threshold functions directly and have strictly non-comparable bounds. WINNOW like LASSO is better for “sparse” linear threshold functions

Part III

Learning with sequences of experts

On-line Learning (Review)

time t	1	2	3	4	\dots	t
expert 1	.5	3	2	1	\dots	$x_{t,1}$
expert 2	.75	-1.5	-1	-2	\dots	$x_{t,2}$
expert 3	-1.5	3	2	-4	\dots	$x_{t,3}$
expert 4	.75	-1.3	1.5	1.5	\dots	$x_{t,4}$
alg. preds	0	1.5	1.5	.75	\dots	\hat{y}_t
label	.75	-1.5	2	1	\dots	y_t
alg. loss	0.56	9	.25	.06	\dots	$(\hat{y}_t - y_t)^2$

For a sequence of examples $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\}$

$$\text{Loss}_A(S) = \sum_{t=1}^{\ell} (\hat{y}_t - y_t)^2$$

Aim: to bound $\text{Loss}_A(S)$ in terms of the loss of the best expert.

$$\text{Loss}_i(S) = \sum_{t=1}^{\ell} (x_{t,i} - y_t)^2$$

Static Relative Loss Bounds

For all sequences of examples S

$$\text{Loss}_A(S) \leq \text{Loss}_i(S) + c \ln n \quad [V90, LW94, HKW98]$$

- The loss of the algorithm is bounded in terms of the best **single** expert.

So far the comparator is “static”

Shifting Expert

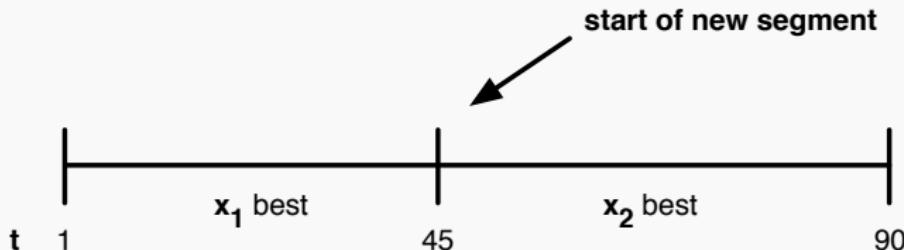
Stream of Examples



- The loss of the algorithm is compared against the loss of the best sequence of experts.
- The loss of a sequence of experts is the sum of the loss of the expert for each segment.
- The algorithms do not know when a new segment begins and which expert are best in each segment.

Problem for Algorithms

The crucial weights may be too small or too large. It might take the algorithm too long to recover.



- Lower bounds on weights [LW, AW, BB]
- Weights as “probabilities” [HWa, V]
- Keep the weights in a bounded region defined in terms of a “divergence function”: [HWb]

Shifting Experts – Details

- In the *static* case the loss bound is given relative to a single *expert*.

The (square) loss defined relative to a sequence of experts

$$\text{Loss}_{\{i_1, i_2, \dots, i_\ell\}}(S) = \sum_{t=1}^{\ell} (x_{t,i_t} - y_t)^2$$

Shifting Experts – Details

- In the *static* case the loss bound is given relative to a single *expert*.
- In the *shifting* case the loss bound is given relative to a *sequence* of experts.

The (square) loss defined relative to a sequence of experts

$$\text{Loss}_{\{i_1, i_2, \dots, i_\ell\}}(S) = \sum_{t=1}^{\ell} (x_{t,i_t} - y_t)^2$$

Algorithm Design

Given the number of distinct sequences of experts is

$$O\left(n \frac{\ell}{k}\right)^k$$

Why?

Using the “experts” algorithm we can design an algorithm with a loss bound of

$$L_A(S) \leq L^* + ck \left(\ln n + \ln \frac{\ell}{k} + \text{“const”} \right)$$

Where $L^* := \text{Loss}_{\{i_1, i_2, \dots, i_\ell\}}(S)$

How?

Caveats:

- Algorithm is exponential in k
- Loss bound dependent on ℓ

Fixed and Variable Share Algorithms

- **Parameters:** $0 \leq \eta$ and $0 \leq \alpha \leq 1$.
- **Initialization:** Set the weights to $w_{1,1}^s = \dots = w_{1,n}^s = \frac{1}{n}$.
- **Prediction:** Let $v_{t,i} = \frac{w_{t,i}^s}{W_t}$, where $W_t = \sum_{i=1}^n w_{t,i}^s$. Predict with $\hat{y}_t = \mathbf{v}_t \cdot \mathbf{x}_t$
- **Loss Update:** After receiving the t th outcome y_t ,

$$\forall i : 1, \dots, n : w_{t,i}^m = w_{t,i}^s e^{-\eta L(y_t, x_{t,i})}$$

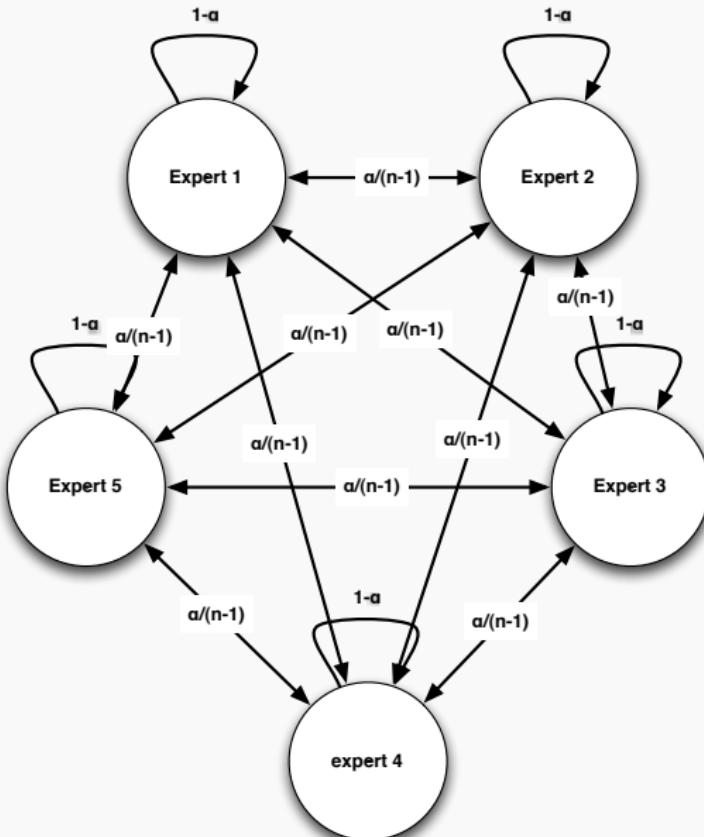
- **Fixed Share Update:**

- $\text{pool} = \sum_{i=1}^n \alpha w_{t,i}^m$
- $\forall i : 1, \dots, n : w_{t+1,i}^s = (1 - \alpha) w_{t,i}^m + \frac{1}{n-1} w_{t,i}^s (\text{pool} - \alpha w_{t,i}^m)$

- **Variable Share Update:**

- $\text{pool} = \sum_{i=1}^n (1 - (1 - \alpha)^{L(y_t, x_{t,i})}) w_{t,i}^m$
- $\forall i : 1, \dots, n : w_{t+1,i}^s = (1 - \alpha)^{L(y_t, x_{t,i})} w_{t,i}^m + \frac{1}{n-1} w_{t,i}^s (\text{pool} - (1 - (1 - \alpha)^{L(y_t, x_{t,i})}))$

Fixed Share – Visualization



Shifting Loss Bounds

Shifting Experts:

- Let

$$k := \text{size}(\{i_1, i_2, \dots, i_\ell\}) := \sum_{k=1}^{\ell-1} [i_k \neq i_{k+1}]$$

- $L^* := \text{Loss}_{\{i_1, i_2, \dots, i_\ell\}}(S)$
- Choose $\alpha \in [0, 1]$,

We then have the bound:

$$L_A(S) \leq L^* + c[(\ell - 1)(H(\alpha^*) + D(\alpha^* \parallel \alpha)) + k \ln(n - 1) + \ln n],$$

where $\alpha^* = \frac{k}{\ell-1}$, $H(p) = \frac{1}{p} \ln \frac{1}{p} + \frac{1}{1-p} \ln \frac{1}{1-p}$ and

$$D(p^* \parallel p) = p^* \ln \frac{p^*}{p} + (1 - p^*) \ln \frac{1 - p^*}{1 - p}$$

When $\alpha = \frac{k}{\ell-1}$, then the above is upper bounded by

$$L_A(S) \leq L^* + c[k \ln \frac{\ell - 1}{k} + k \ln(n - 1) + \ln n + k],$$

Shifting Loss Bounds (Proof Sketch – 1)

Proof Sketch – 1

1. Observe that the bound $\text{Loss}_A(S) \leq \text{Loss}_i(S) + c \ln n$ for the expert setting trivially generalises to

$$\text{Loss}_A(S) \leq \text{Loss}_i(S) + c \ln \frac{1}{w_{1,i}} \quad (i \in [n])$$

i.e., rather than assign all experts uniform probability initially we can choose an arbitrary prior weights (probability).

2. Consider the following set of ℓ^n initial weights:

$$w_{1,\{i_1, i_2, \dots, i_\ell\}} := \frac{1}{n} (1 - \alpha)^{\ell-1-k} \left(\frac{\alpha}{n-1} \right)^k \quad (\{i_1, i_2, \dots, i_\ell\} \in [n]^\ell)$$

3. Observe that the sum of these weights is 1.
4. These are just “meta-experts” that describe all possible expert sequences over ℓ trials.

Shifting Loss Bounds (Proof Sketch – 2)

Proof Sketch – 2

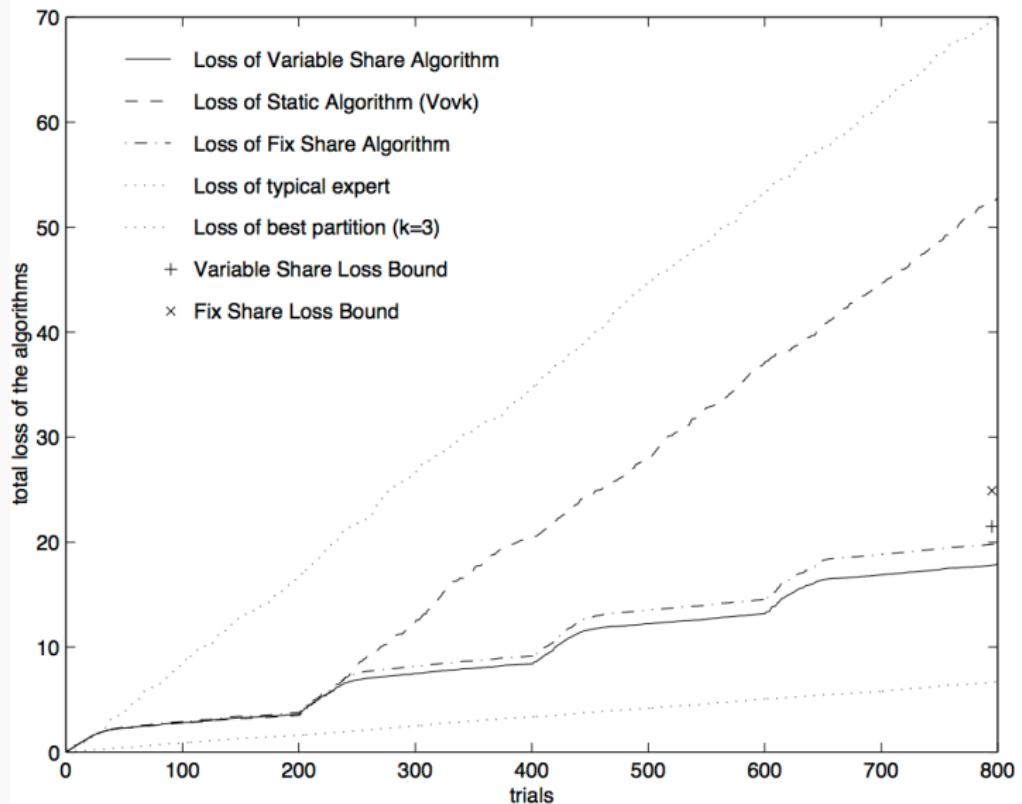
1. Observe if we apply the WA algorithm with these meta-experts then the previous bound follows (with some algebra) by just simplifying $\ln \frac{1}{w_{1,\{i_1, i_2, \dots, i_\ell\}}}.$
2. With a more detailed analysis one can then show FIXED-SHARE algorithm perfectly simulates the prediction of the algorithm with ℓ^n meta-experts in $O(n)$ time.

□

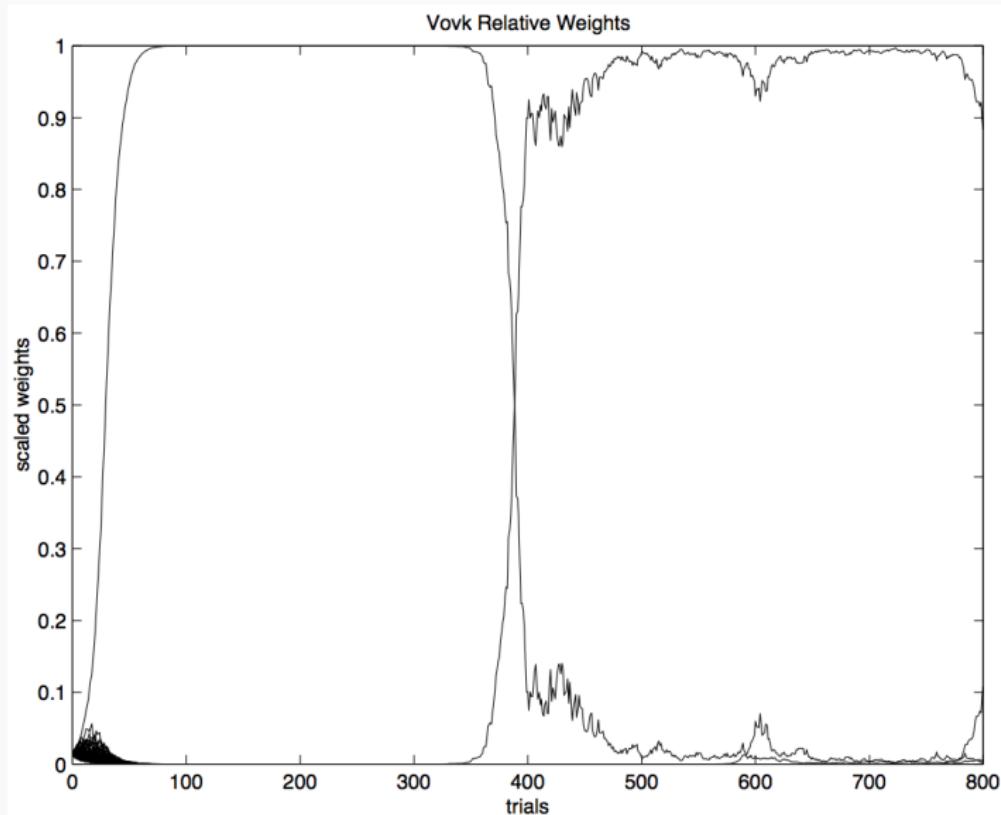
Experiment (simulation)

1. Trials (ℓ) : 800
2. Shifts(k): 3
3. Number of Experts (n) : 64
4. Loss function (L): $L(y, \hat{y}) = (y - \hat{y})^2$ ($c = 1/2, \eta = 2$)
5. Share Parameter(α) : 0.024
6. Typical expert expected Loss/Trial : 1/12
7. “Best” expert expected Loss/Trial : 1/120

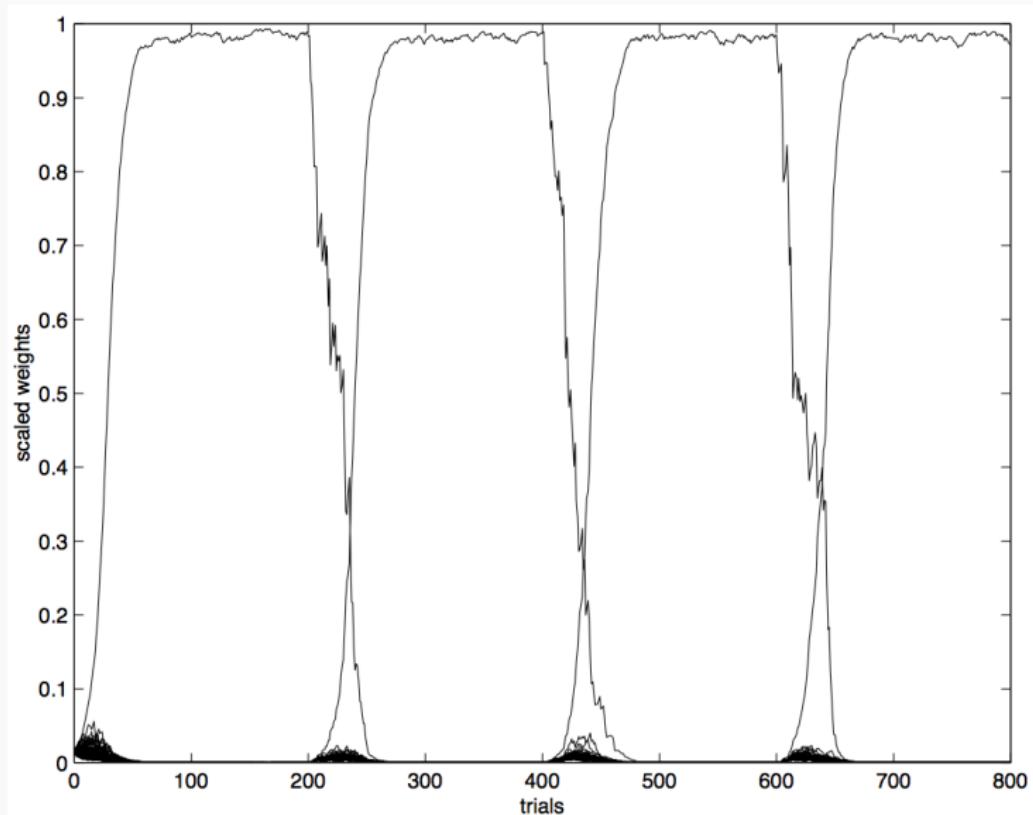
Share Algorithms – Performance



Expert Algorithm's Weights



Variable Share Weights



Some Applications

1. Predicting disk idle times
2. Online load balancing (process migration determination)
3. Predicting TCP packet inter-arrival times
4. Financial prediction by combining portfolios
5. Combining language for domain and topic adaptation

Problems – 1

1. Suppose $\mathcal{X} = \{\text{TRUE}, \text{FALSE}\}^n$. Give a polynomial time algorithm \mathcal{A} with a mistake bound $M(\mathcal{A}) \leq O(n^2)$ for any example sequence which is consistent with a k -literal conjunction. Your answer should contain an argument that $M(\mathcal{A}) \leq O(n^2)$.
2. State the perceptron convergence theorem [Novikoff] explaining the relation with the hard margin support vector machine solution.
3. **[Hard]:** Define the c -regret of learning algorithm as

$$c\text{-regret}(m) = L_{\mathcal{A}}(S) - c \min_{i \in [n]} L_i(S)$$

thus the usual regret is the 1-regret.

- 3.1 Argue for the weighted majority set-up argue that without randomised prediction it is impossible for all training sequences to obtain sublinear c -regret for $c < 2$.
- 3.2 Show how to select β to obtain sublinear 2-regret.
4. Consider binary prediction with expert advice, with a perfect expert. Prove that any algorithm makes at least $\Omega(\min(m, \log_2 n))$ mistakes in the worst case.

Problems – 2

1. Recall that by tuning the weighted majority we achieved a bound

$$M \leq 2.63 \min_i M_i + 2.63 \ln n$$

Now by using randomisation in the prediction, design an algorithm with a bound that has the property

$$\frac{M}{m} \leq \min_{i \in [n]} \frac{M_i}{m} \text{ as } m \rightarrow \infty,$$

for the weighted majority setting (i.e., the mean prediction error of the algorithm is bounded by the mean prediction error of the “best” expert). Recalling that m is the number of examples (and the “tuning” of the algorithm may depend on m). For contrast compare this to problem 3.1 above.

Useful references

1. Nicolò Cesa-Bianchi and Gábor Lugosi, *Prediction, learning, and games.*, (2006), Note this is a book.
2. N. Littlestone, *Learning quickly when irrelevant attributes abound: a new linear threshold algorithm*, (1988).
3. N. Littlestone and M. K. Warmuth. *The weighted majority algorithm*, (1994)
4. V. Vovk, *Aggregating strategies*, (1990).
5. Y. Freund and R. Schapire, *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*, (1997)
6. Haussler, D., Kivinen, J. and Warmuth, M.K. *Sequential Prediction of Individual Sequences Under General Loss Functions*, (1998)
7. M. Herbster and M. Warmuth, *Tracking the Best Expert*, (1998)
8. J. Kivinen and M. Warmuth, *Averaging Expert predictions*, (1999)
9. S. Shalev-Schwartz, *Online Learning and Online Convex Optimization*, (2011)

6. Graphs in Machine Learning (Graph-based Semi-supervised learning)

COMP0078: Supervised Learning

Mark Herbster

15 November 2021

University College London
Department of Computer Science
SL-GM-21v1

Today's Plan

Graph-based Semi-Supervised Learning

- Motivating Semi-Supervised Learning
- The Graph Laplacian
- Spectral Clustering
- Graph-based Semi-supervised Learning
- Understanding the Laplacian Interpolation (effective resistance)
- Generalising the Laplacian (p -Laplacian)

Part I

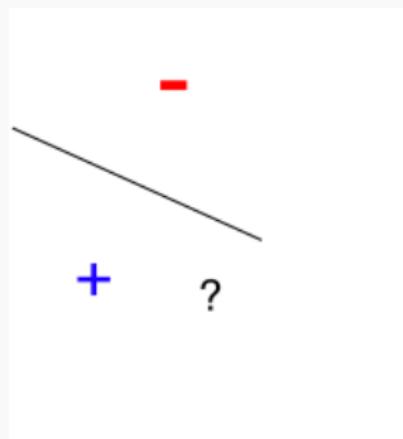
Motivation

“To learn from both labeled and unlabeled data”

Motivation (Learning)

The hypothesis of SSL is that the structure of the input space may be exploited to improve learning for example.

- **[Cluster hypothesis]:** If points are in the same cluster, they are likely to be of the same class.
- **[Manifold hypothesis]:** The (high-dimensional) data lie on a low-dimensional manifold.

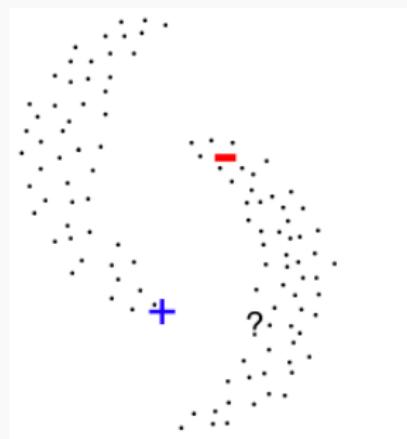


“To learn from both labeled and unlabeled data”

Motivation (Learning)

The hypothesis of SSL is that the structure of the input space may be exploited to improve learning for example.

- **[Cluster hypothesis]:** If points are in the same cluster, they are likely to be of the same class.
- **[Manifold hypothesis]:** The (high-dimensional) data lie on a low-dimensional manifold.



Thought problem

Problem

There exists two Gaussian distributions and a hyperplane. Points are generated by either distribution and labelled according to which side of the hyperplane they fall.

Will unlabelled data help?

1. No known relationship between the hyperplane and the inputs (“ x ”’s).
2. The hyperplane is known to separate the “centers” of the gaussians.

Motivation (Data)

Unlabeled may be “cheap” wrt to labeled the data. For example,

- We have a collection of news stories – yet only a few are labeled with respect to content
- We have a great deal of collected voice (eg siri, google voice); transcription slow and expensive
- In drug design we have many sample molecules, each experiment to test for certain properties, may require days of labwork and considerable expense.

Recall: Supervised learning

- Typical goal
 - Given data (pattern,target) $\mathcal{S} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\}$ infer a function f such that $f(\mathbf{x}_i) \approx y_i$ for future data $\mathcal{S}' = \{(\mathbf{x}_{\ell+1}, y_{\ell+1}), (\mathbf{x}_{\ell+2}, y_{\ell+2}), \dots\}$.
 - Classification : $y \in \{-1, +1\}$; Regression : $y \in \mathbb{R}$
- Algorithms
 1. Linear Regression
 2. Neural Networks
 3. Decision Trees
 4. Support Vector Machines

Recall: Unsupervised learning

- Typical goal
 - Given data (patterns) $\mathcal{S} = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\}$
Model the data.
 - For example (clustering)
Give a partition/function of $f : \mathcal{S} \rightarrow \{1, \dots, k\}$ of \mathcal{S}
Such that $f(\mathbf{x}_i) = f(\mathbf{x}_j) \Rightarrow \mathbf{x}_i \approx \mathbf{x}_j$
 $f(\mathbf{x}_i) \neq f(\mathbf{x}_j) \Rightarrow \mathbf{x}_i \not\approx \mathbf{x}_j$
- Algorithms
 1. Clustering (k-means)
 2. Dimensionality Reduction (PCA)

Semi-supervised Learning

- Typical goal
 - Given data $\mathcal{S} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell), \mathbf{x}_{\ell+1}, \dots, \mathbf{x}_n\}$
infer a function f such that $f(\mathbf{x}_i) \approx y_i$
- Idea
 - Combine supervised and unsupervised learning.
- Algorithms
 1. Co-Training
 2. Graph-based Semi-Supervised Learning [today]
 3. Semi-supervised Support Vector Machines

Inductive and transductive semi-supervised learning

Definition: Inductive semi-supervised learning

Given a training sample,

$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}, (\mathbf{x}_j)_{j=\ell+1}^n\}$$

an *inductive* learner learns a function $f : X \rightarrow Y$, with the aim that " $f(x) \approx y$ ", **for all**, $x \in X$.

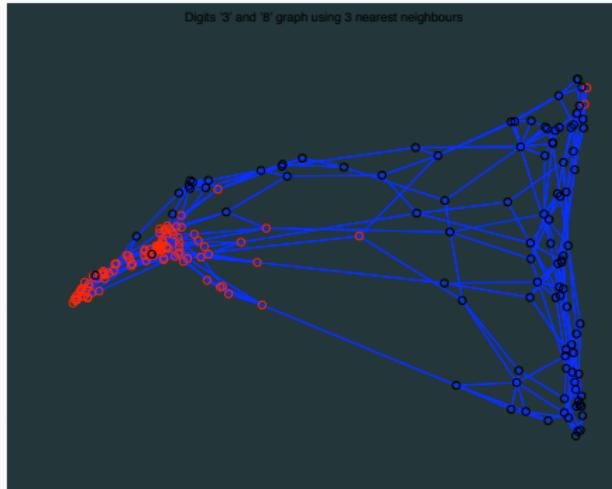
Definition: Transductive semi-supervised learning

Given a training sample,

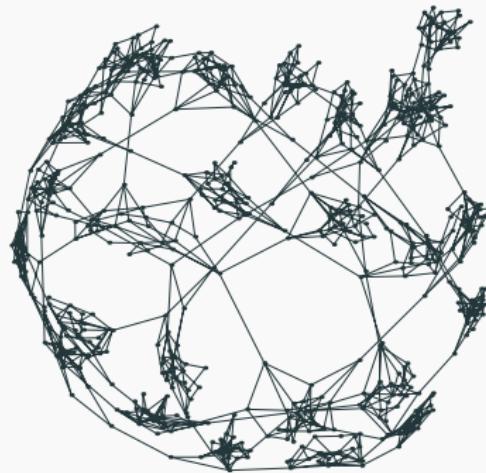
$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}, (\mathbf{x}_j)_{j=\ell+1}^n\}$$

a transductive learner learns a function " $f(x) \approx y$ " only on the **unlabelled** data $f : X_U \rightarrow Y$ (with $X_U = (\mathbf{x}_j)_{j=\ell+1}^n$).

Graph Examples – 1

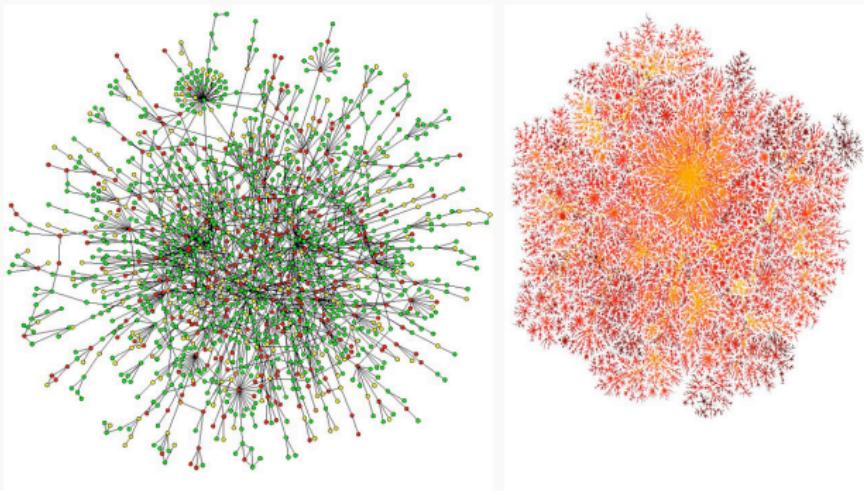


USPS digits 3 and 8



Random graph $G^2_{k_out}(26; 2)$

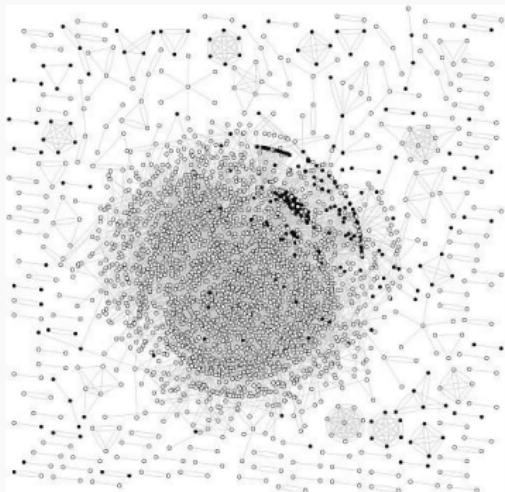
Graph Examples – 2



Yeast protein network

Internet hosts

Graph Examples – 3



Web Spam



Twitter Social Network

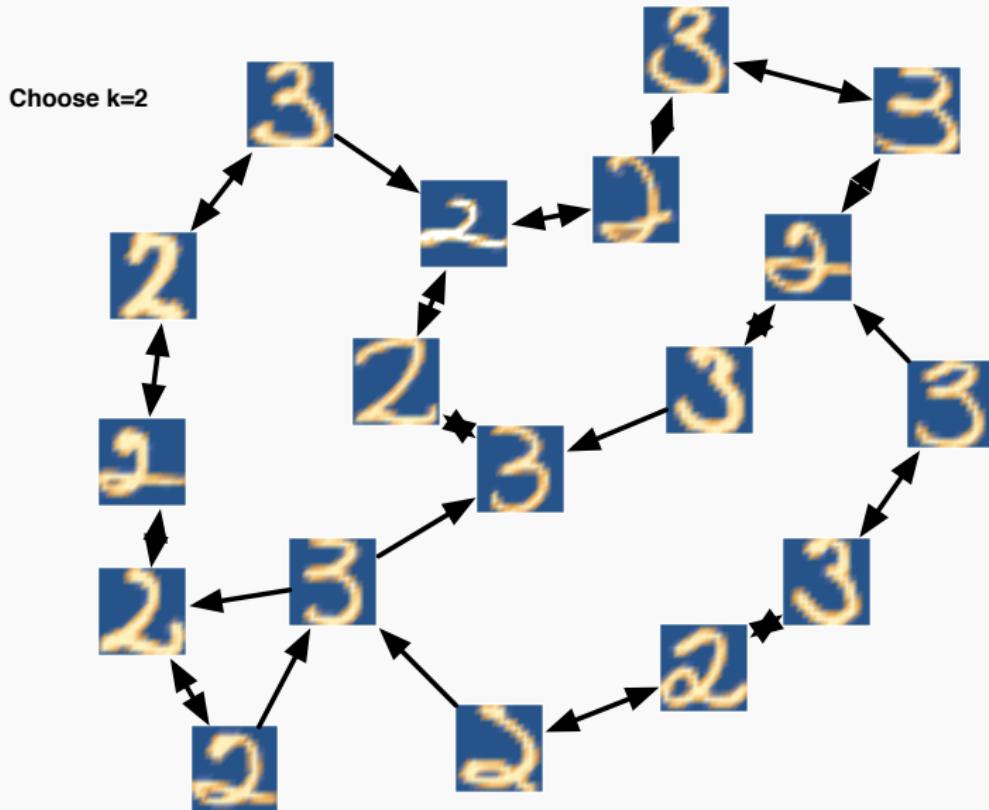
Building graphs

Where do graphs come from?

- Intrinsic: (protein interaction network, social network)
- Built (using a similarity metric $d(p, q)$):
 1. k -Nearest neighbors: Each vertex connects to k nearest neighbors
 2. ϵ -ball: Every point is connected to every other within ϵ distance
 3. Tree-based: Build a MST ("Minimum spanning tree") or a SPT ("Shortest paths tree")
 4. Weighted graph: The edge between p and q is weighted $e^{-cd(p,q)}$
 5. Combo: The first three methods may be combined with weights.

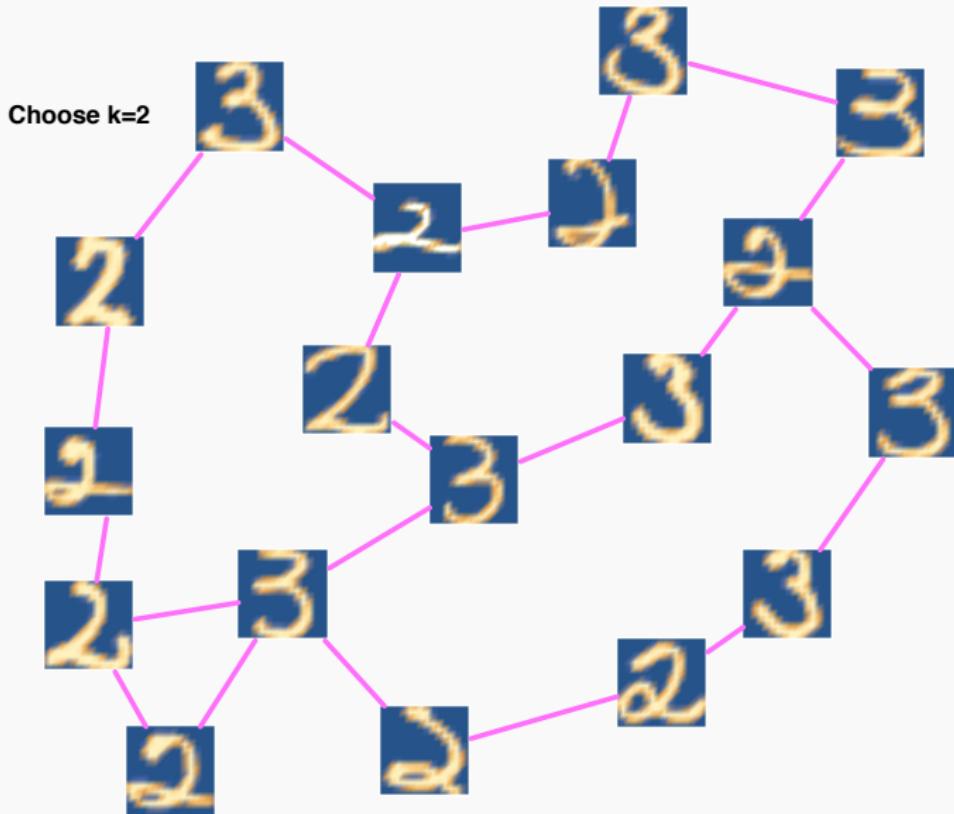
Comment: The quality of predictions depend on the quality of the graph. A continuing area of research.

Example 2-nn (Step 1)



A 2-nn graph (step 1)

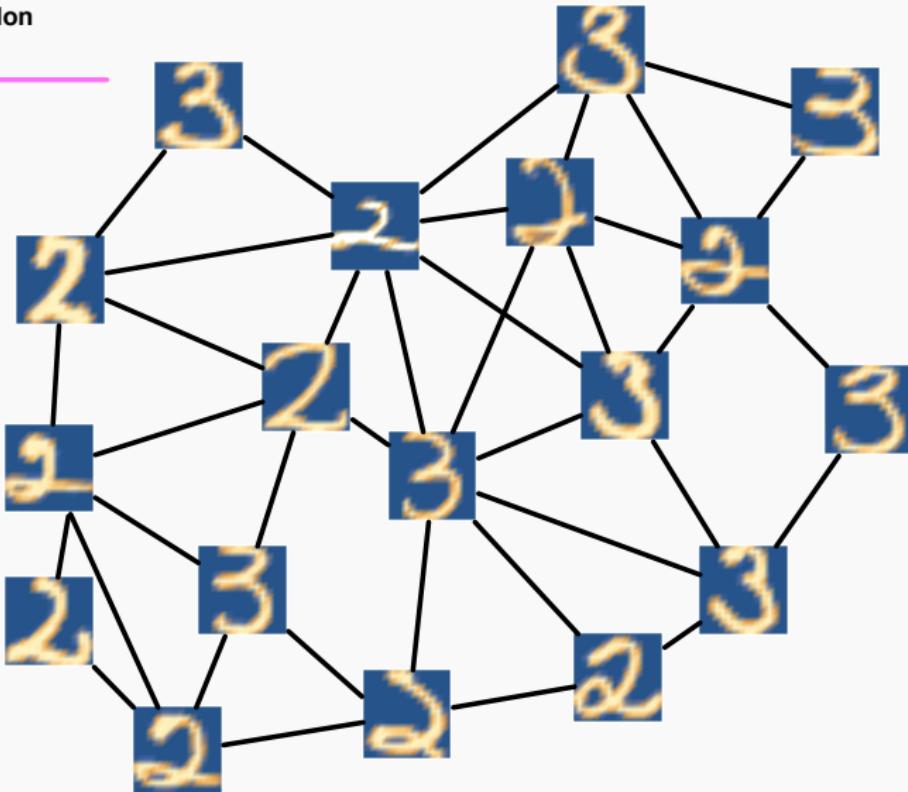
Example 2-nn (Step 2)



A 2-nn graph (step 2)

Example ϵ -ball

Choose Epsilon



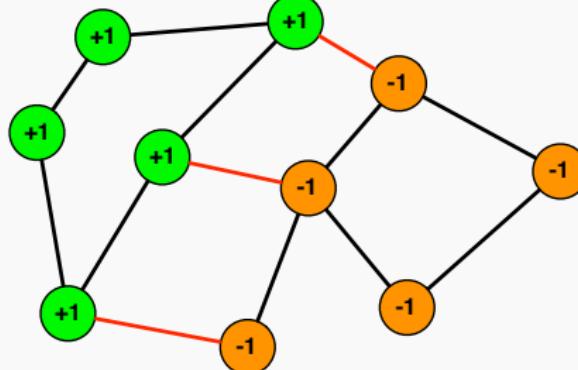
An ϵ -ball graph

Part II

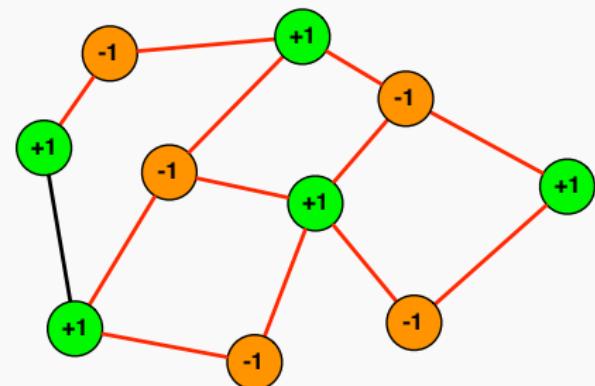
The graph Laplacian

Complexity of a Graph “Classifiers”

- A “Complexity” $\Phi_G(\mathbf{u})$ is associated with a labeling $\mathbf{u} \in \{-1, 1\}^n$



$$\Phi_G(\mathbf{u}) = 3$$



$$\Phi_G(\mathbf{u}) = 12$$

The graph Laplacian (matrix) will allow to extend this discrete measure of complexity/smoothness to real values.

The Graph Laplacian

Definition

The graph Laplacian is $\mathbf{L}(G) := \mathbf{D} - \mathbf{W}$ where \mathbf{W} is the (symmetric non-negatively weighted) adjacency matrix and $\mathbf{D} := \text{diag}(d_1, \dots, d_n)$ is diagonal matrix of (weighted) degrees where $d_v := \sum_{i \neq v} w_{vi}$.

Associated (semi) inner product and (semi) norm

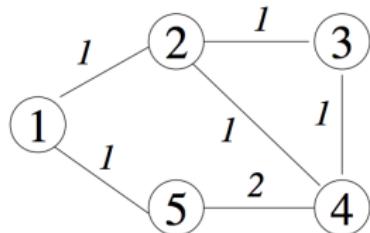
- The laplacian $\mathbf{L}(G)$ is positive semi-definite.
- Define

$$\begin{aligned}\langle \mathbf{u}, \mathbf{v} \rangle_G &:= \mathbf{u}^\top \mathbf{L} \mathbf{v} \\ \|\mathbf{u}\|_G^2 &:= \langle \mathbf{u}, \mathbf{u} \rangle_G = \sum_{(i,j) \in E(G)} w_{ij} (u_i - u_j)^2\end{aligned}$$

- Define $\mathbb{S}_n := \{\mathbf{u} : \sum_{i=1}^n u_i = 0\}$; assume G is connected
- For $\mathbf{u} \in \mathbb{R}^n$ ($\mathbf{u} \in \mathbb{S}_n$) we've defined (semi) norm and a (semi)inner product.
- Observe that the constant vector $\mathbf{1}$ is the eigenvector with smallest eigenvalue=0.
- In a connected graph $\mathbf{1}$ is the only eigenvector with eigenvalue 0.
- The associated kernel: $\mathbf{K}_G(i, j) = \mathbf{e}_i^\top \mathbf{L}^+ \mathbf{e}_j = L_{ij}^+$ (pseudoinverse)
- Observe that $\phi : [n] \rightarrow \mathbb{S}_n$ where $\phi(i) := \mathbf{e}_i^\top \mathbf{L}^+$. is feature for the kernel \mathbf{L}^+ .
- Checking

$$\langle \phi(i), \phi(j) \rangle = \langle \mathbf{e}_i^\top \mathbf{L}^+, \mathbf{e}_j^\top \mathbf{L}^+ \rangle_G = \mathbf{e}_i^\top \mathbf{L}^+ \mathbf{L} \mathbf{e}_j^\top \mathbf{L}^+ = \mathbf{e}_i^\top \mathbf{L}^+ \mathbf{L} \mathbf{L}^+ \mathbf{e}_j = \mathbf{e}_i^\top \mathbf{L}^+ \mathbf{e}_j = L_{ij}^+$$

Laplacian Example



$$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 3 & -1 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & -1 & -1 & 4 & -2 \\ -1 & 0 & 0 & -2 & 3 \end{pmatrix}$$

Figure 1. A Graph on five vertices and its Laplacian matrix. The weights of edges are indicated by the numbers next to them. All edges have weight 1, except for the edge between vertices 4 and 5 which has weight 2.

Example from *Algorithms, Graph Theory, and Linear Equations in Laplacian Matrices* (D. Spielman 2010)

The *weights* on the edges represent the strength of a connection. A larger weight indicates a strong connection. If the weight is 0 there is then no edge.

Checking that $\mathbf{u}^\top L \mathbf{u} = \sum_{i < j} w_{ij}(u_i - u_j)^2$

$$\begin{aligned}\|\mathbf{u}\|_G^2 &:= \langle \mathbf{u}, \mathbf{u} \rangle_G \\&= \mathbf{u}^\top (D - W) \mathbf{u} \\&= \sum_{i,j} D_{ij} u_i u_j - \sum_{i,j} w_{ij} u_i u_j \\&= \sum_i d_i u_i^2 - 2 \sum_{i < j} w_{ij} u_i u_j \\&= \sum_i (\sum_{j \neq i} w_{ij}) u_i^2 - 2 \sum_{i < j} w_{ij} u_i u_j \\&= 2 \sum_{i < j} w_{ij} (u_i^2) - 2 \sum_{i < j} w_{ij} u_i u_j \\&= \sum_{i < j} w_{ij} (u_i^2 + u_j^2) - 2 \sum_{i < j} w_{ij} u_i u_j \\&= \sum_{i < j} w_{ij} (u_i - u_j)^2\end{aligned}$$

Part III

Spectral Clustering

Spectral Clustering

To divide the graph into natural clusters without labels.

Objectives

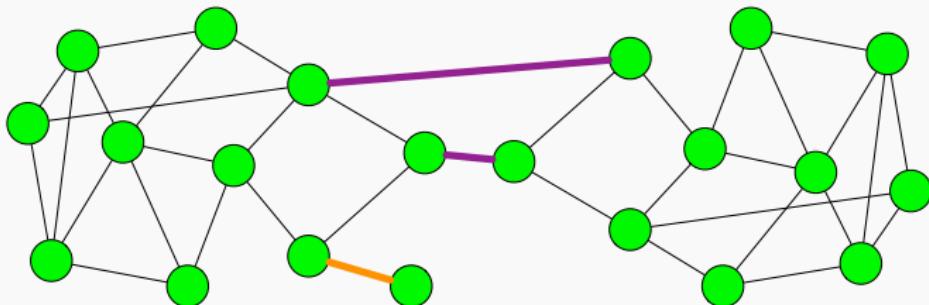
Mincut:

$$\text{cut}(A, B) = \sum_{i \in A, j \in B} w_{ij}$$

RatioCut:

$$\text{RatioCut}(A, B) = \text{cut}(A, B) \left(\frac{1}{|A|} + \frac{1}{|B|} \right)$$

Example



The **Mincut** and the **RatioCut**.

Observations

1. The Mincut can be computed efficiently in $O(n^3)$ time
2. The RatioCut is **NP-hard**
3. The mincut is not useful for clustering as it does not capture the idea of balanced clusters unlike RatioCut.
4. Both objectives can be generalised to K class case.
5. A third objective called *normalised cuts* is also used commonly in practice.

Approximated in much the same way as the BalancedCut but with a “normalised Laplacian” for space reasons we omit.

- Since we cannot compute exactly and efficiently we will **approximate!**

Relaxing Balanced Cuts – 1

For simplicity we restrict the problem before relaxing.

Problem: Perfectly Balanced Cuts

$$\min_{A,B:|A|=|B|} \text{cut}(A, B)$$

Still NP-Hard.

We will represent a clustering by a vector $\mathbf{u} \in \{-1, 1\}^n$ so that

$$u_i := \begin{cases} 1 & i \in A \\ -1 & i \in B \end{cases}$$

Observe that now,

$$\text{cut}(A, B) = \frac{1}{4} \sum_{i < j} w_{ij} (u_i - u_j)^2 = \frac{1}{4} \mathbf{u}^\top \mathbf{L} \mathbf{u}$$

and that

$$|A| = |B| \implies \sum_{i=1}^n u_i = 0 \implies \mathbf{u} \perp \mathbf{1}$$

Perfectly Balanced Cuts (Objective rewritten)

$$\min_{\mathbf{u}} \mathbf{u}^\top \mathbf{L} \mathbf{u} \text{ s.t. } \mathbf{u} \in \{-1, 1\}^n, \quad \mathbf{u} \perp \mathbf{1}, \quad \|\mathbf{u}\|^2 = n$$

The term $\|\mathbf{u}\|^2 = n$ is redundant. (Why?) we introduce as it will be useful for our relaxation of the problem.

Relaxing Balanced Cuts – 2

Relaxed Balanced Cuts (Objective rewritten)

$$\min_{\mathbf{u}} \mathbf{u}^T \mathbf{L} \mathbf{u} \text{ s.t. } \mathbf{u} \in \mathbb{R}^n, \quad \mathbf{u} \perp \mathbf{1}, \quad \|\mathbf{u}\|^2 = n \quad (1)$$

Relaxing \mathbf{u} to the reals, leads to an efficiently solvable problem. The key is the following *variational characterisation* of eigenvalues.

Generalised Rayleigh-Ritz

If $\lambda_1 \leq \dots \leq \lambda_n$ are the eigenvalues of a real-valued symmetric matrix \mathbf{M} and $\mathbf{v}_1, \dots, \mathbf{v}_n$ are the corresponding eigenvectors then we have the following,

$$\begin{aligned} \lambda_k &= \min_{\substack{\mathbf{u} \neq 0 \\ \mathbf{u} \perp \mathbf{v}_1, \dots, \mathbf{v}_{k-1}}} \frac{\mathbf{u}^T \mathbf{M} \mathbf{u}}{\mathbf{u}^T \mathbf{u}} : \mathbf{u} \perp \mathbf{v}_1, \dots, \mathbf{v}_{k-1} \\ &= \min_{\substack{\mathbf{u} \neq 0 \\ \mathbf{u}^T \mathbf{u} = 1}} \mathbf{u}^T \mathbf{M} \mathbf{u} : \mathbf{u} \perp \mathbf{v}_1, \dots, \mathbf{v}_{k-1}, \quad \mathbf{u}^T \mathbf{u} = 1 \end{aligned}$$

Observe therefore that since $\mathbf{1}$ is the first eigenvector of the Laplacian that the second eigenvector (up to scaling by a constant) is then the solution of (1).

Relaxing Balanced Cuts – 3

In fact the second eigenvector is also a relaxed solution to balanced cuts without the constraint $|A| = |B|$. Recall,

$$\text{RatioCut}(A, B) = \text{cut}(A, B) \left(\frac{1}{|A|} + \frac{1}{|B|} \right)$$

Now instead represent a clustering by a vector \mathbf{u} so that

$$u_i := \begin{cases} \sqrt{\frac{|B|}{|A|}} & i \in A \\ -\sqrt{\frac{|A|}{|B|}} & i \in B \end{cases}$$

Observe that now,

$$\mathbf{u}^\top \mathbf{L} \mathbf{u} = \left(\frac{|B|}{|A|} + \frac{|A|}{|B|} + 2 \right) \text{cut}(A, B) = (|A| + |B|) \text{RatioCut}(A, B),$$

and thus with \mathbf{u} relaxed then (1) is also the solution to relaxed RatioCut.

In practice

We have a real-valued solution we need to convert to a discrete solution. Simplest is to use sign u in practice there are a number of heuristics that tend to work better based around finding a "natural gap" in the "middle" of the range u_i values. Likewise we may wish to extend the objective to include K classes. See [L07] for details.

Part IV

The graph Laplacian

Building graph classifiers

Algorithmic frameworks

- Minimum cut
- Laplacian
- p -Laplacian
- Kernel (Laplacian-based) [Not discussed, many graph kernels serve as alternatives to \mathbf{L}^+ .]
- Markov Random Field (Ising Model) [Not discussed]

Recall: Regularization

As we have defined a “complexity” (here a semi-norm), we may now immediately apply regularization.

Idea

We obtain our predictor $f^* \in \mathcal{F}$,

$$f^* := \operatorname{argmin}_{f \in \mathcal{F}} \text{error}(\text{"training data"}, f) + \lambda \text{complexity}(f)$$

Example: Ridge regression

$$\mathbf{w}^* := \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^m (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \lambda \sum_{\ell=1}^n w_\ell^2 \equiv (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^\top \mathbf{w}$$

Interpolation

Today we simplify and focus on the **interpolation** case.

1. We assume there exists at least one $f^* \in \mathcal{F}$ that predicts perfectly on the training examples.
2. We select the zero-error f^* with minimum complexity.

Alternately,

$$f_\lambda := \operatorname{argmin}_{f \in \mathcal{F}} \text{error}(\text{"training data"}, f) + \lambda \operatorname{complexity}(f)$$

and set

$$f^* := \lim_{\lambda \rightarrow 0} f_\lambda$$

Part V

Minimum cut transduction

Minimum-cut

The minimum cut method:

- For each edge we may also introduce weights $(w_e)_{e \in E(\mathcal{G})} \subset [0, \infty)$
- Given a set of labeled vertices $(v \in V(\mathcal{G}), y \in \{-1, 1\})$

$$\{(v_{i_1}, y_1), (v_{i_2}, y_2), \dots (v_{i_\ell}, y_\ell)\}$$

- The hypothesis vector is given by

$$\hat{\mathbf{u}}_t = \arg \min_{\mathbf{u} \in \{-1, 1\}^n} \left\{ \sum_{(i,j) \in E(\mathcal{G})} w_{ij} |u_i - u_j| : u_{i_1} = y_1, \dots, u_{i_\ell} = y_\ell \right\}$$

- This may be solved by a “min-cut”/“max-flow” algorithm in cubic time or by linear programming.
- Introduced by Blum and Chawla in 2001. First graph-based SSL algorithm
- Inspired many other algorithms but not popular in practice since it has tendency to produce unbalanced labelings. The same issue as using “min-cut” as an objective for clustering.

Minimum cut (Observations)

Observations

- The objective's minima is unchanged when relaxed to real values,

$$\min_{\mathbf{u} \in \{-1,1\}^n} \left\{ \sum_{(i,j) \in E(\mathcal{G})} w_{ij} |u_i - u_j| : u_{i_1} = y_1, \dots, u_{i_\ell} = y_\ell \right\} =$$
$$\min_{\mathbf{u} \in \mathbb{R}^n} \left\{ \sum_{(i,j) \in E(\mathcal{G})} w_{ij} |u_i - u_j| : u_{i_1} = y_1, \dots, u_{i_\ell} = y_\ell \right\}$$

- The problem is *ill-posed* (no unique solution). E.g., how should we label v_3 ,



Part VI

Laplacian-based transduction

Laplacian-based

Laplacian-based method:

- Given a weighted graph with labeled vertices ($y \in \{-1, 1\}$)

$$\{(v_{i_1}, y_1), (v_{i_2}, y_2), \dots (v_{i_\ell}, y_\ell)\}$$

- The interpolant vector is given by

$$\bar{\mathbf{u}}_t = \arg \min_{\mathbf{u} \in \mathbb{R}^n} \left\{ \sum_{(i,j) \in E(\mathcal{G})} w_{ij} |u_i - u_j|^2 : u_{i_1} = y_1, \dots, u_{i_\ell} = y_\ell \right\}$$

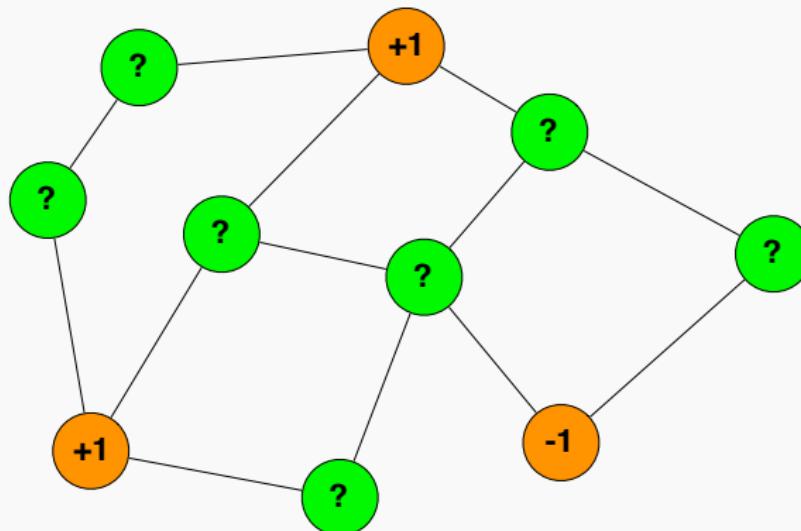
- To predict at vertex v use $\hat{u}_v := \text{sign}(\bar{u}_v)$.
- May be optimized in cubic time by solving a linear system of equations.
- It is known by many names not limited to *harmonic minimization*, *label propagation*, *Laplacian interpolated regularization*.
- Variations and embellishments on the above account for a large portion of Graph-based SSL. This method has performed well on a large variety dataset with respect to other SSL methods. See for example [BMN04] and [ZGL03].

Optimisation by Consensus for Laplacian interpolation

- The computation is determined by a network of nodes.
- Each nodes' computation is
 1. Simple
 2. Local (depends only on neighbors)
 3. Parallel
 4. Asynchronous
- Fault tolerant
- Two types of nodes fixed and free
- Network topology identified with topology of data

Consensus Algorithm (Relaxation)

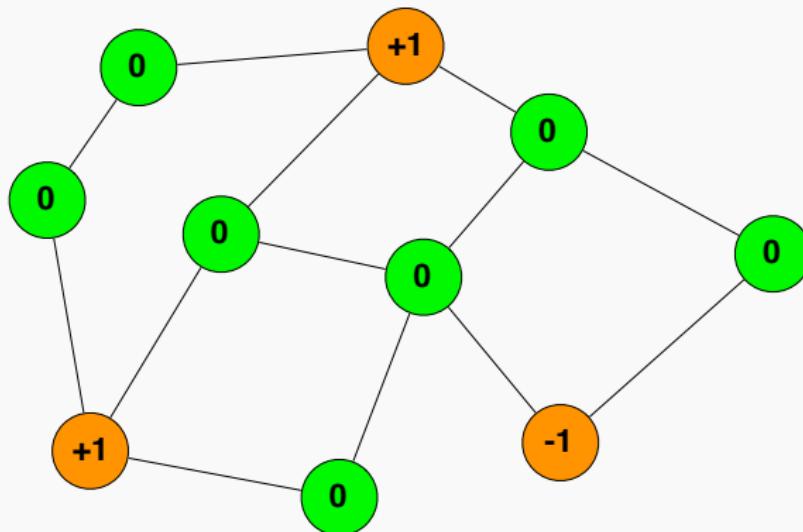
1. Select a free vertex i (Green) uniformly at random
2. Average neighbors $u'(i) = \frac{\sum_{j \in \Gamma(i)} u_j}{d_i}$
3. Goto 1



- For a weighted graph use a weighted mean (via W)

Consensus Algorithm (Relaxation)

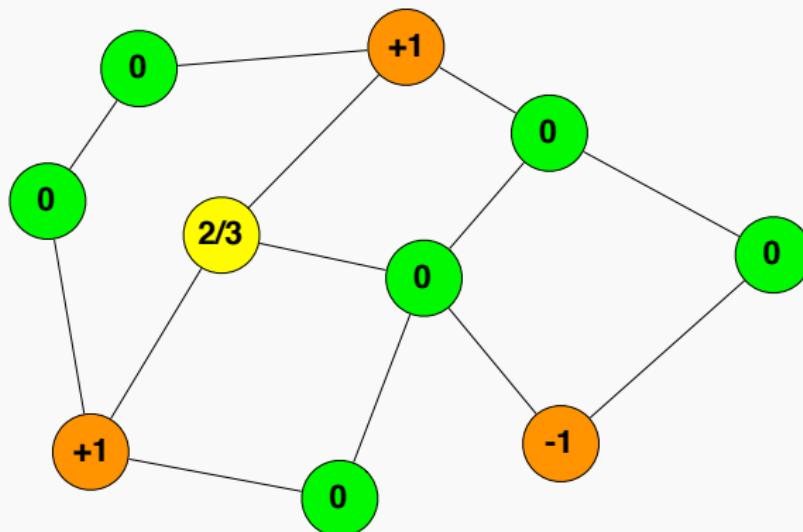
1. Select a free vertex i (Green) uniformly at random
2. Average neighbors $u'(i) = \frac{\sum_{j \in \Gamma(i)} u_j}{d_i}$
3. Goto 1



- For a weighted graph use a weighted mean (via W)

Consensus Algorithm (Relaxation)

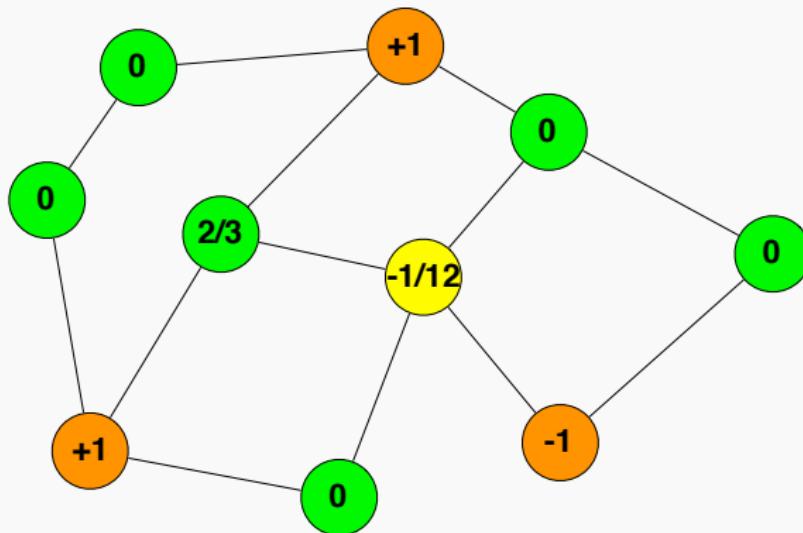
1. Select a free vertex i (Green) uniformly at random
2. Average neighbors $u'(i) = \frac{\sum_{j \in \Gamma(i)} u_j}{d_i}$
3. Goto 1



- For a weighted graph use a weighted mean (via W)

Consensus Algorithm (Relaxation)

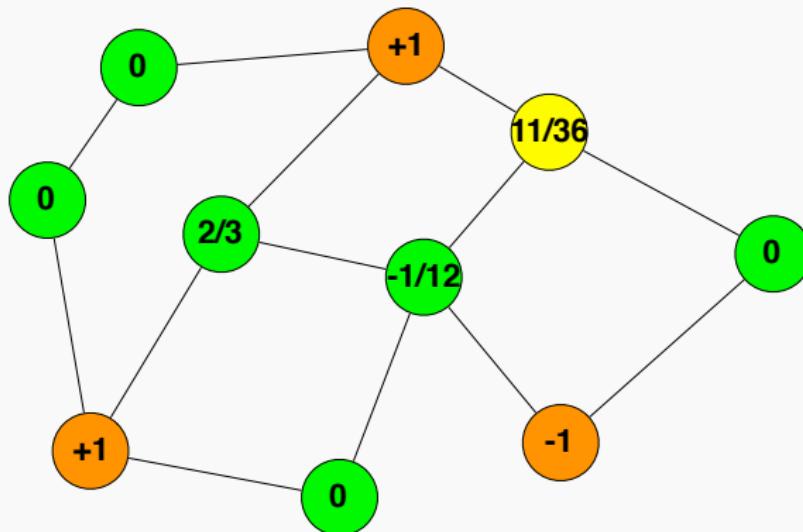
1. Select a free vertex i (Green) uniformly at random
2. Average neighbors $u'(i) = \frac{\sum_{j \in \Gamma(i)} u_j}{d_i}$
3. Goto 1



- For a weighted graph use a weighted mean (via W)

Consensus Algorithm (Relaxation)

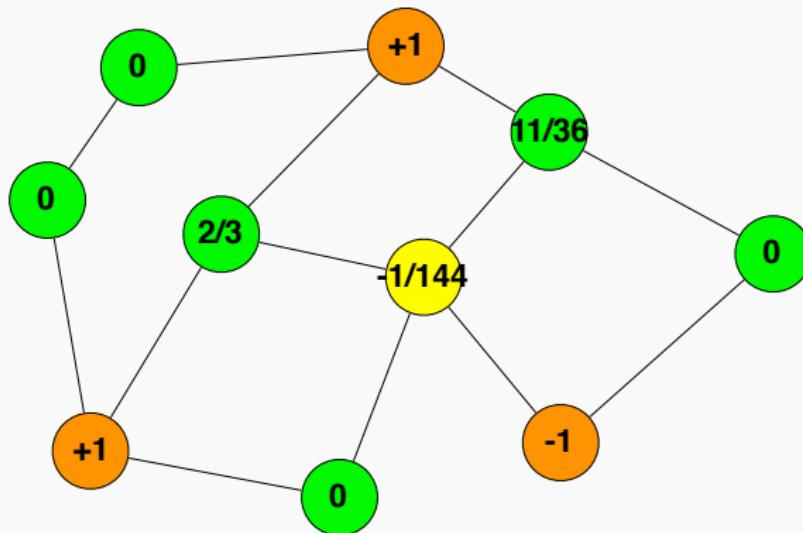
1. Select a free vertex i (Green) uniformly at random
2. Average neighbors $u'(i) = \frac{\sum_{j \in \Gamma(i)} u_j}{d_i}$
3. Goto 1



- For a weighted graph use a weighted mean (via W)

Consensus Algorithm (Relaxation)

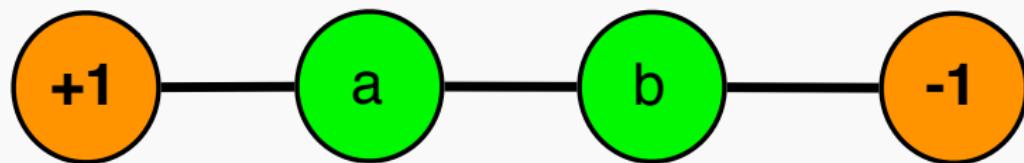
1. Select a free vertex i (Green) uniformly at random
2. Average neighbors $u'(i) = \frac{\sum_{j \in \Gamma(i)} u_j}{d_i}$
3. Goto 1



- For a weighted graph use a weighted mean (via W)

Four Node Problem

Exercise: Consider the graph,



1. What do nodes *a* and *b* converge to?
2. Why? Give justification.

Consensus Convergence Proof (Step 1)

Sketch

1. Define “energy” of a labelling u as the squared-norm induced by the Laplacian
2. Denote \mathbf{u} before selection after selection \mathbf{u}'
3. Observe $\|\mathbf{u}\|^2 \geq \|\mathbf{u}'\|^2 \geq \|\mathbf{u}''\|^2 \geq \dots \geq 0$ **Why?!?**
4. Hence convergence in “energy”

Proof – Comments

Exercise (difficult [technical]):

- Suppose \bar{u} denotes the minimum.
- Claim: $u^{(t)} \rightarrow \bar{u}$ as $t \rightarrow \infty$.

Observations

Theorem (harmonic solution)

$$\bar{\mathbf{u}}_t = \arg \min_{\mathbf{u} \in \mathbb{R}^n} \left\{ \sum_{i < j} w_{ij} |u_i - u_j|^2 : u_{i_1} = y_1, \dots, u_{i_\ell} = y_\ell \right\}$$

implies

$$\bar{u}_i = \frac{\sum_{j=1}^n w_{ij} u_j}{d_i} \quad (i = \ell + 1, \dots, n)$$

$$\bar{u}_i = y_i \quad (i = 1, \dots, \ell)$$

- Consensus solves of a Linear System
- Consensus is a type of “coordinate” descent
- Many “efficient” methods to solve linear systems
- Why consensus?
 1. Simple
 2. Parallel
- Matrix form of consensus called “label propagation”

Part VII

Interpreting Laplacian-based transduction

Explanations

Why does it work? Types of explanations? Justifications?

1. As a resistive network
2. As a random walk on a network
3. Under certain conditions the graph-Laplacian may approximate the Riemannian Laplacian of the data Manifold
4. Via various predictive performance guarantees

Connectivity: Is the shared idea in “1” and “2”. The densities of inputs points are reflected in the connectivity of the built graph, which modifies the “*ambient*” metric of the inputs.

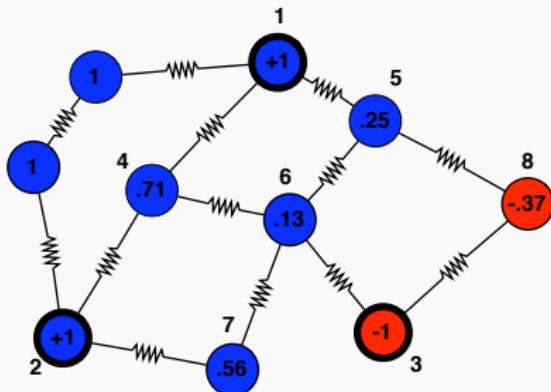
Graph to resistive network

- Identify graph with a resistive network
- Labels are voltage constraints; weights on edges correspond to inverse resistances.
- Power of a labeling

$$P(\mathbf{u}) := \|\mathbf{u}\|_{\mathcal{G}}^2 = \sum_{(i,j) \in E(\mathcal{G})} w_{ij} |u_i - u_j|^2 = \mathbf{u}^\top L \mathbf{u}$$

- Label by minimizing power

$$\bar{\mathbf{u}} = \arg \min_{\mathbf{u} \in \mathbb{R}^n} \{P(\mathbf{u}) : u_{i_1} = y_1, \dots, u_{i_\ell} = y_\ell\}$$

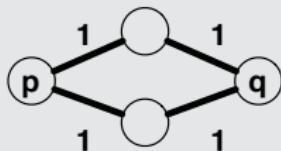


Effective Resistance – 1

Definition

For a pair of vertices the effective resistance between these vertices is the resistance of the total network when a voltage source is connected across them, i.e., it is the voltage difference needed to induce a *unit* current flow between a pair of vertices.

Example



- Geodesic distance is 2 from v_p to v_q
- Effective resistance is 1 from v_p to v_q

Effective Resistance – 2 (Kirchoff's Circuit Laws)

Junction rule

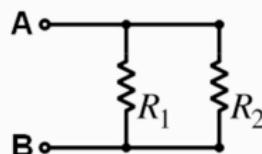
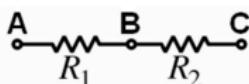
$$\sum I_{in} = \sum I_{out}$$

Effective resistance

$$R_{eff} = \frac{V_{tot}}{I_{tot}}$$

Loop rule

$$\sum V_{loop} = 0$$



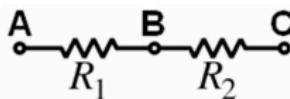
Effective Resistance – 3 (Series Circuit)

Series circuit:

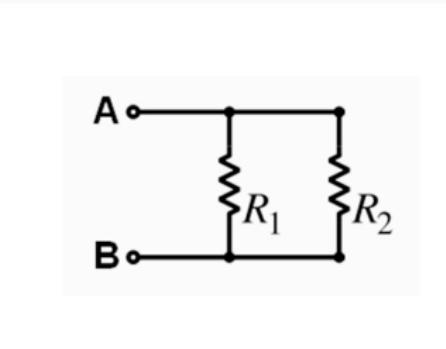
$$\begin{aligned}V_{tot} &= V_1 + V_2 \\I_{tot} &= I_1 = I_2 \\R_{eff} &= \frac{V_1 + V_2}{I_{tot}} \\&= \frac{V_1}{I_1} + \frac{V_2}{I_2} \\&= R_1 + R_2\end{aligned}$$

Generalises to:

$$R_{eff} = R_1 + R_2 + \dots + R_n$$



Effective Resistance – 4 (Parallel Circuit)



Parallel circuit:

$$\begin{aligned}V_{tot} &= V_1 = V_2 \\I_{tot} &= I_1 + I_2 \\R_{eff} &= \frac{V_{tot}}{I_1 + I_2} \\&= \frac{V_1}{\frac{V_1}{R_1} + \frac{V_2}{R_2}} \\&= \frac{V_1}{V_1 \left(\frac{1}{R_1} + \frac{1}{R_2} \right)} \\ \Rightarrow \frac{1}{R_{eff}} &= \frac{1}{R_1} + \frac{1}{R_2}\end{aligned}$$

Generalises to:

$$\frac{1}{R_{eff}} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4}$$

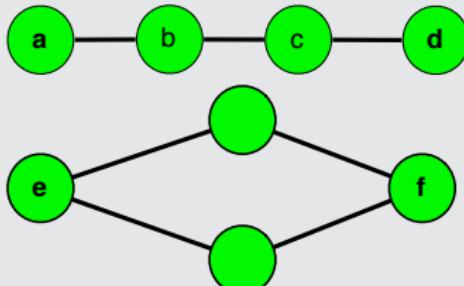
Effective Resistance – 6

(Computation via optimization): One may prove that

$$r_G(p, q) = \frac{1}{\min_{\mathbf{u} \in \mathbb{R}^n} \{ \|\mathbf{u}\|_G^2 : u_p = 1, u_q = 0 \}}$$

Exercise

1. Compute the effective resistance between a and d
2. Compute the effective resistance between a and c
3. Compute the effective resistance between e and f
4. Challenge: Use non-unit resistors on the edges



Effective Resistance – 7 (Kernel)

In fact the **effective resistance** is the squared distance induced by the kernel \mathbf{L}^+ .

Theorem [KR93]

Effective resistance between v_p and v_q is

$$r_{\mathbf{G}}(p, q) = L_{pp}^+ + L_{qq}^+ - 2L_{pq}^+$$

Proof – 1

Lemma

Let $\mathbf{x} \in \mathcal{H}$ then

$$\|\mathbf{x}\|^{-2} = \min_{\mathbf{w} \in \mathcal{H}} \{\|\mathbf{w}\|^2 : \langle \mathbf{w}, \mathbf{x} \rangle = 1\}.$$

Proof Of Lemma

Observe that $\frac{\mathbf{x}}{\|\mathbf{x}\|^2}$ is a feasible solution to the minimization. Suppose $\mathbf{v} = \frac{\mathbf{x}}{\|\mathbf{x}\|^2} + \alpha \mathbf{z}$ is a feasible solution with $\|\mathbf{v}\|^2 < \|\mathbf{x}\|^{-2}$. Since $\langle \mathbf{v}, \mathbf{x} \rangle = 1$ then $\langle \mathbf{z}, \mathbf{x} \rangle = 0$. Hence

$$\|\mathbf{v}\|^2 = \|\mathbf{x}\|^{-2} + 2\alpha \|\mathbf{x}\|^{-2} \langle \mathbf{z}, \mathbf{x} \rangle + \|\mathbf{z}\|^2 = \|\mathbf{x}\|^{-2} + \|\mathbf{z}\|^2$$

which contradicts the supposition.

Proof – 2

Proof of [KR93]

Recall,

$$r_{\mathbf{G}}(p, q) = \frac{1}{\min_{\mathbf{u} \in \mathbb{R}^n} \{\|\mathbf{u}\|_{\mathbf{G}}^2 : u_p = 1, u_q = 0\}},$$

Observe that

$$\begin{aligned} \min_{\mathbf{u} \in \mathbb{R}^n} \{\|\mathbf{u}\|_{\mathbf{G}}^2 : u_p = 1, u_q = 0\} &= \min_{\mathbf{u} \in \mathbb{R}^n} \{\|\mathbf{u}\|_{\mathbf{G}}^2 : u_p - u_q = 1\} \\ &= \min_{\mathbf{u} \in \mathbb{S}_n} \{\|\mathbf{u}\|_{\mathbf{G}}^2 : u_p - u_q = 1\} \\ &= \min_{\mathbf{u} \in \mathbb{S}_n} \{\|\mathbf{u}\|_{\mathbf{G}}^2 : \langle \mathbf{e}_p^\top \mathbf{L}^+ - \mathbf{e}_q^\top \mathbf{L}^+, \mathbf{u} \rangle_{\mathbf{G}} = 1\} \end{aligned}$$

The first two equalities follow from the fact that any $c \neq 0$ that $\|\mathbf{u}\|_{\mathbf{G}}^2 = \|\mathbf{u} + c\mathbf{1}\|_{\mathbf{G}}^2$ and the third follows since

$$\langle \mathbf{e}_p^\top \mathbf{L}^+, \mathbf{u} \rangle_{\mathbf{G}} = \mathbf{e}_p^\top \mathbf{L}^+ \mathbf{L} \mathbf{u} = u_p$$

for any $\mathbf{u} \in \mathbb{S}_n$. Now using the Lemma on the final equality we have

$$r_{\mathbf{G}}(p, q) = \|\mathbf{e}_p^\top \mathbf{L}^+ - \mathbf{e}_q^\top \mathbf{L}^+\|_{\mathbf{G}}^2 = (\mathbf{e}_p^\top \mathbf{L}^+ - \mathbf{e}_q^\top \mathbf{L}^+) \mathbf{L} (\mathbf{e}_p^\top \mathbf{L}^+ - \mathbf{e}_q^\top \mathbf{L}^+)^{\top} = L_{pp}^+ + L_{qq}^+ - 2L_{pq}^+.$$

□

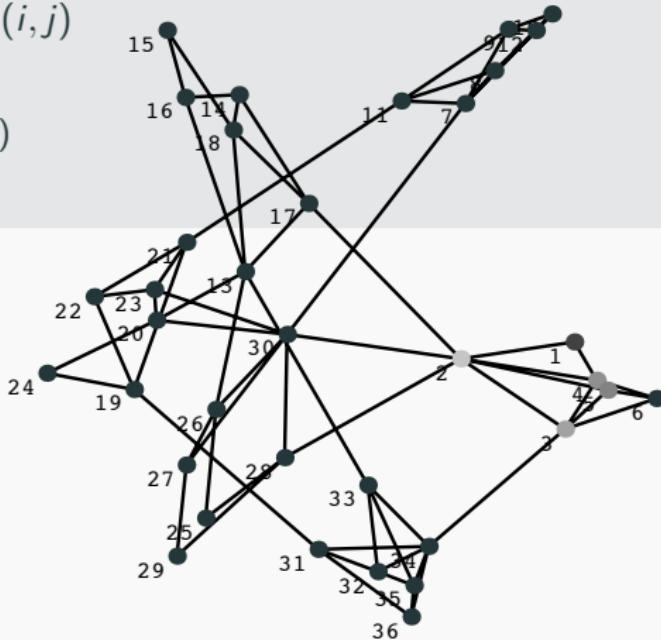
Effective resistance 8 – (Graph connectivity)

The kernel L^+ , “contains” the following connectivity information.

Inverse Connectivities

- Graph : $R_{\text{tot}} := \sum_{i>j} r_G(i,j)$
- Vertex :

1. $R(v) := \sum_{i=1}^n r_G(v, i)$
2. $L_{vv}^+ = \frac{R(v)}{n} - \frac{R_{\text{tot}}}{n^2}$

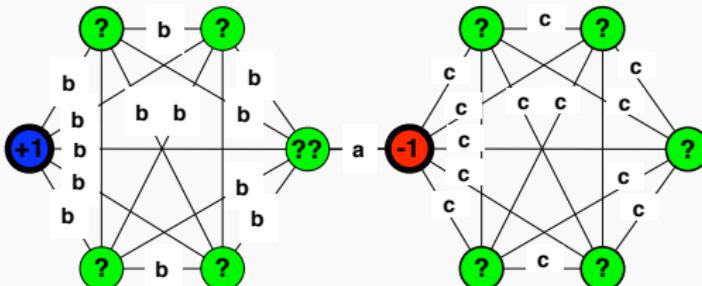


Grey-scale L_{vv}^+ : $L_{30,30}^+ = .21$ (min), $L_{15,15}^+ = .94$ (max)

The “Intuition”

From “distance” to “resistance”

The base global “distance” $d(p, q)$ is *adapted* according to the empirical distances via the effective resistance $r(p, q)$.



Two $m - \text{cliques}$ connected by an edge with distances $a < b$

- What is the label at “??”
- Effective resistance between vertices within a clique whose edge-resistance are $\leq d$ is $r_{m\text{-clique}} \leq \frac{2d}{m}$
- If $\frac{2b}{m} < a$ then label “+1”
- Conclusion: labelings respect cluster structure

Random walk on graphs – 1

Definition

The **transition probability** P_{ij} of going from vertex i to vertex j on a “step” is

$$P_{ij} := \frac{w_{ij}}{\sum_i w_{ij}}$$

where W is the symmetric weighted adjacency matrix (L is the associated Laplacian).

Example

Consider the following path graph with unit weights on edges. What is the probability that we reach vertex labeled “1” before we reach the vertex labeled “0”?



Random walk on graphs – 2

Observation

Let p_i denote the probability that from vertex i we reach the vertex labeled “1” before we reach the vertex labeled “0.”

Claim

For vertices v_2, \dots, v_5 we have $p_i = \frac{p_{i-1} + p_{i+1}}{2}$. Why?



Observe that the boundary conditions are met for vertices v_1 and v_6 .

Random walk on graphs – 3

Theorem

Given an unweighted graph G with adjacency matrix W , a set of vertices V_0 labeled “0”, and a set vertices V_1 labeled “1” and let p_i denote the probability that a random walker started at vertex i reaches a vertex in V_1 before reaching a vertex in V_0 . Then

$$\mathbf{p} = \arg \min_{\mathbf{p} \in \mathbb{R}^n} \left\{ \sum_{i < j} w_{ij} |p_i - p_j|^2 : (p_k = 0)_{v_k \in V_0}, (p_k = 1)_{v_k \in V_1} \right\}$$

Proof. The equations and the boundary conditions that determine the random walk are isomorphic to the boundary conditions and equations of label propagation.

Summary. The label propagation solution has natural interpretations in terms of voltages and random walks.

Random walk on graphs – 4

We also have the following result connecting effective resistance to commute time.

Definition

The commute time $c_G(i,j)$ between vertex i and vertex j is the *expected* time to travel from i to j and back again.

Theorem

The commute time $c_G(i,j)$ between i and j is twice the total degree times the effective resistance.

$$c_G(i,j) := r_G(i,j) \sum_{i,j}^n w_{ij} .$$

Part VIII

Generalising Laplacian-based transduction

Overview

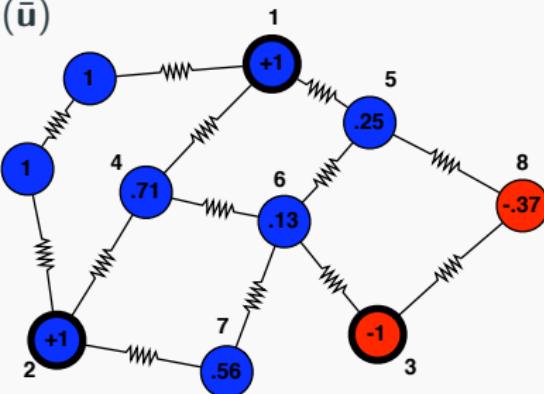
- We generalize the Laplacian norm to a p -Laplacian norm
- Motivation
 1. Aim to control the sparsity of the cut in analogy to lasso
 2. Tradeoff the weighting of cut-size verse geodesic distance
- A generalized p -network theory follows from a definition of p -energy
- When $p = 1$ we have a “pipe” network, $p = 2$ an electrical network,
 $p \rightarrow \infty$ “shortest path network”

Generalising to the p -Laplacian

- Identify graph with a p -resistive network
- Labels are voltage constraints.
- Redefine energy (power) as

$$P_p(\mathbf{u}) := \|\mathbf{u}\|_{\mathcal{G},p}^p = \sum_{(i,j) \in E(\mathbf{G})} w_{ij} |u_i - u_j|^p$$

- Label by minimizing energy
 $\bar{\mathbf{u}} = \arg \min_{\mathbf{u} \in \mathbb{R}^n} \{P_p(\mathbf{u}) : u_{i_1} = y_1, \dots, u_{i_\ell} = y_\ell\}$
- Predict with $\text{sign}(\bar{\mathbf{u}})$



Properties of p -Resistive Networks

Analogues of usual “electric network” theory now follow ...

1. Kirchoff's laws
2. Ohm's law
3. Conservation of energy principle
4. Rayleigh's monotonicity principle
5. Black box principle (2-port)
6. The “rules” of resistors in parallel and in series
7. Triangle Inequality

We will derive some of these.

Kirchoff's current law – 1

Definitions

1. Edge Resistance : $\pi_{ij} = \frac{1}{w_{ij}}$
2. Voltage constraints : $\mathcal{S} := (u_{i_1} = y_1, \dots, u_{i_\ell} = y_\ell)$
3. Constrained vertices : $V_{\mathcal{S}} := \{i_1, \dots, i_\ell\}$.
4. (Key Definition) : Energy with respect to \mathcal{S}

$$P(\mathcal{S}) := \min_{\mathbf{u} \in \mathbb{R}^n} \{P(\mathbf{u}) : u_{i_1} = y_1, \dots, u_{i_\ell} = y_\ell\}. \quad (2)$$

Hence at the minimum we have

$$\begin{aligned} \frac{\partial P(\mathbf{u})}{\partial u_i} &= 0 & v_i \notin V_{\mathcal{S}} \\ \sum_{j:j \sim i} \frac{|u_i - u_j|^{p-1} \operatorname{sgn}(u_i - u_j)}{\pi_{ij}} &= 0 & v_i \in V_{\mathcal{S}}. \end{aligned} \quad (3)$$

Kirchoff's current law – 2

Definitions

The *current* from vertex v_i to v_j of a network

$$I_{ij}(\mathcal{S}) := \frac{|u_i - u_j|^{p-1} \operatorname{sgn}(u_i - u_j)}{\pi_{ij}} \quad (4)$$

if $p = 2$ this is *Ohm's law*.

The *net current* from vertex v_i is defined as

$$I_i := \sum_{j:j \sim i} I_{ij}.$$

We see that (3) is *Kirchoff's current law* for \mathbf{I}

$$0 = I_i \quad v_i \notin V_S \quad (5)$$

I.e., the conservation of flow follows from the definition of energy (2).

p -resistance

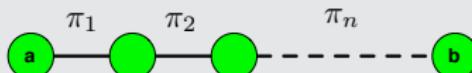
Definition

The (effective) p -resistance between any two vertices a and b is

$$r_p(a, b) = \left[\min_{\mathbf{u} \in \mathbb{R}^n} \{P_p(\mathbf{u}) : u_a = 1, u_b = 0\} \right]^{-1}.$$

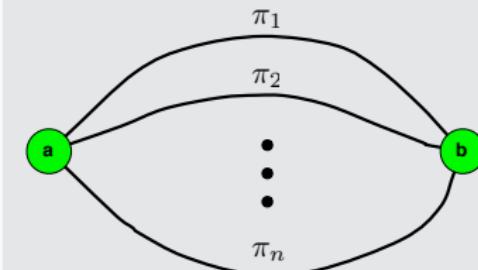
Resistors in series

$$r_{G,p}(a, b) = \left(\sum_{i=1}^n \pi_i^{\frac{1}{p-1}} \right)^{p-1}$$



Resistors in parallel

$$r_{G,p}(a, b) = \left(\sum_{i=1}^n \frac{1}{\pi_i} \right)^{-1}$$



Note if $p = 1$ then $r_{G,1}(a, b) = \frac{1}{\text{"mincut between } a \text{ and } b\text{"}}$

p -resistance trade-off

Idea : We can gain insight to behaviour of interpolation in a normed space by understanding the metric induced by the norm. The p -resistance is the metric induced by the p -Laplacian norm. To understand the metric, we consider the simplified cases corresponding to the serial and parallel laws.

Recall p -norm

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^n x_i^p \right)^{\frac{1}{p}} \quad p \in [1, \infty)$$

Observation

1. For resistors in series the “law” is $(\frac{1}{p-1})$ -norm on the edge resistances.
2. For resistors in parallel the “law” is -1 – “pseudo-norm” on the edge resistances (triangle inequality fails).
3. The parallel law is independent of p .

Comparing p -resistances

The key cases

- [$p = 1$] Serial law is a ∞ -norm (pipe network) – max of resistances
- [$p = 2$] Serial law is a 1-norm (electric network) – sum of resistances
- [$p = \infty$] Serial law is a 0-“pseudo-norm” (geodesic network) – number of resistors

p -Resistive Triangle inequality [technical]

Theorem

Given a graph \mathcal{G} and vertices v_a , v_b , and v_c then

$$r_{\mathcal{G},p}(a, c) \leq \left(r_{\mathcal{G},p}(a, b)^{\frac{1}{p-1}} + r_{\mathcal{G},p}(b, c)^{\frac{1}{p-1}} \right)^{p-1} \quad p \in [1, \infty)$$

thus

$$r_{\mathcal{G},1}(a, c) \leq \max(r_{\mathcal{G},1}(a, b), r_{\mathcal{G},1}(b, c)) \quad (p = 1)$$

p -Resistance interpretation

- Generalises the mincut ($p = 1$) and harmonic solution ($p = 2$)
- As $p \rightarrow \infty$ solution directly connected to geodesic path length ([technical])
- More generally the effective resistance *emphasizes* balances “connectivity” ($p \rightarrow 1$) with geodesic path length as $p \rightarrow \infty$
- The case $p = 2$ is convenient from an optimization perspective and likewise can combine nicely kernel techniques

Part IX

Mistake Bounds for interpolation with the p -Laplacian

Model : predicting the labeling of a graph

- Aim: learn a function $\mathbf{u} : V \rightarrow \{-1, +1\}$ corresponding to a labeling of a graph $\mathcal{G} = (V, E)$ and $V = \{1, \dots, n\}$.
 - Learning proceeds in trials
- for** $t = 1, \dots, \ell$ **do**
1. Nature selects $v_t \in V$
 2. Learner predicts $\hat{y}_t \in \{-1, +1\}$
 3. Nature selects $y_t \in \{-1, +1\}$
 4. If $\hat{y}_t \neq y_t$ then mistakes = mistakes + 1
- Learner's goal: *minimize* mistakes
 - Bound: mistakes $\leq f(\text{complexity}(\mathbf{u}), \text{structure}(\mathcal{G}))$

Minimum p -Seminorm Interpolation

Recall that our algorithm works as follows.

Minimum p -seminorm Interpolation

Input: $\{(i_t, y_t)\}_{t=1}^\ell \subseteq \{1, 2, \dots, n\} \times \{-1, 1\}$.

for $t = 1, \dots, \ell$ **do**

Receive query: $i_t \in \{1, \dots, n\}$

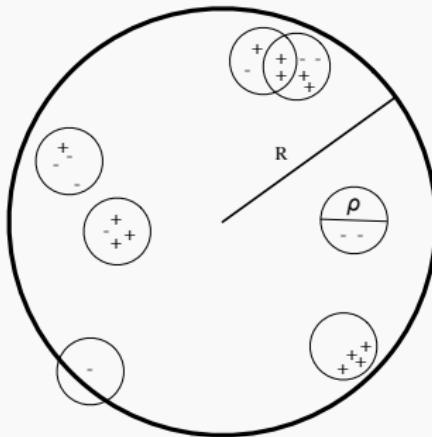
Predict: $\hat{y}_{i_t} = \text{sign}(\mathbf{e}_{i_t}^\top \mathbf{w}_t) = \text{sign}(w_{t,i_t})$

Receive label: y_{i_t}

if $\hat{y}_{i_t} = y_{i_t}$ **then** mistakes = mistakes + 1

$\mathbf{w}_{t+1} = \arg \min_{\mathbf{u} \in \mathbb{R}^n} \{P_p(\mathbf{u}) : u_{i_1} = y_{i_1}, \dots, u_{i_t} = y_{i_t}\}$

Covering



Input space X of radius R with cover number $\mathcal{N}_\rho = 7$.

- Bounds to be dependent on structure of input space X .
- Novikoff is only dependent on X through radius R .
- Expectation is that a typical ambient input space is only sparsely populated (cf manifold/cluster hypotheses).
- Algorithm will depend on the **cover** of X .
- In particular the number of balls \mathcal{N}_ρ of diameter ρ .

Mistake Bound

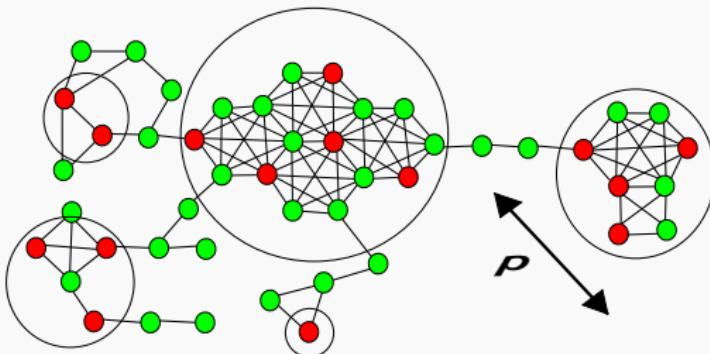
Theorem

Given $\rho \in (1, 2]$ then

$$M \leq \mathcal{N}_\rho + \frac{[\rho \times P_\rho(\mathbf{u})]^{\frac{2}{p}}}{p - 1}$$

for all $0 < \rho$, and for all consistent $\mathbf{u} \in \mathbb{R}^n$.

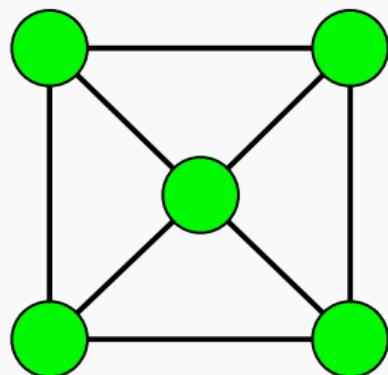
- Relative to any resistive cover w.r.t. r_p
- Both number of covering sets \mathcal{N}_ρ and resistance ρ



Wide Diameter (Tuning)

Definition

A graph has wide diameter (k, Δ_k) if there exist k edge disjoint paths of length no more than Δ_k between all pairs of vertices.



Connectivity: $k = 1$, Wide Diameter: $\Delta_1 = 2$

Connectivity: $k = 2$, Wide Diameter: $\Delta_2 = 2$

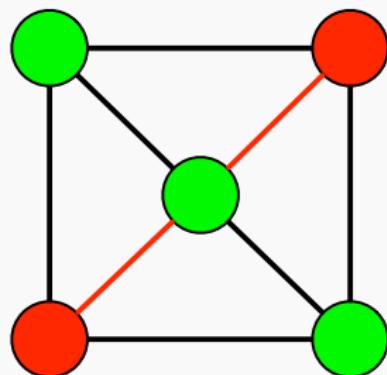
Connectivity: $k = 3$, Wide Diameter: $\Delta_3 = 3$

- The wide diameter is used as a basis for upper bounding p -resistance with the resistors in parallel and series laws.

Wide Diameter (Tuning)

Definition

A graph has wide diameter (k, Δ_k) if there exist k edge disjoint paths of length no more than Δ_k between all pairs of vertices.



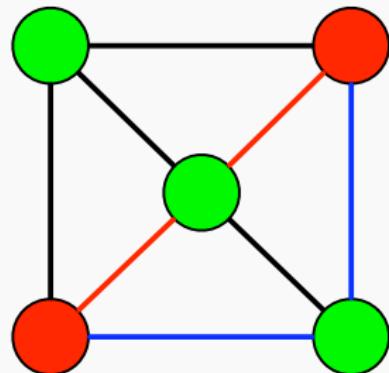
Connectivity: $k = 1$, Wide Diameter: $\Delta_1 = 2$

- The wide diameter is used as a basis for upper bounding p -resistance with the resistors in parallel and series laws.

Wide Diameter (Tuning)

Definition

A graph has wide diameter (k, Δ_k) if there exist k edge disjoint paths of length no more than Δ_k between all pairs of vertices.



Connectivity: $k = 1$, Wide Diameter: $\Delta_1 = 2$

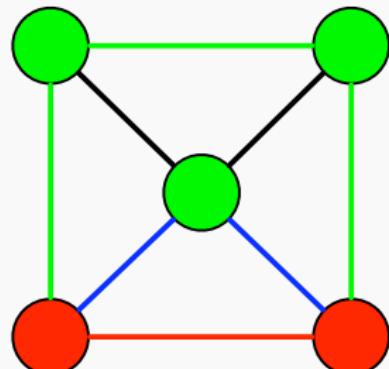
Connectivity: $k = 2$, Wide Diameter: $\Delta_2 = 2$

- The wide diameter is used as a basis for upper bounding p -resistance with the resistors in parallel and series laws.

Wide Diameter (Tuning)

Definition

A graph has wide diameter (k, Δ_k) if there exist k edge disjoint paths of length no more than Δ_k between all pairs of vertices.



Connectivity: $k = 1$, Wide Diameter: $\Delta_1 = 2$

Connectivity: $k = 2$, Wide Diameter: $\Delta_2 = 2$

Connectivity: $k = 3$, Wide Diameter: $\Delta_3 = 3$

- The wide diameter is used as a basis for upper bounding p -resistance with the resistors in parallel and series laws.

Graph prediction

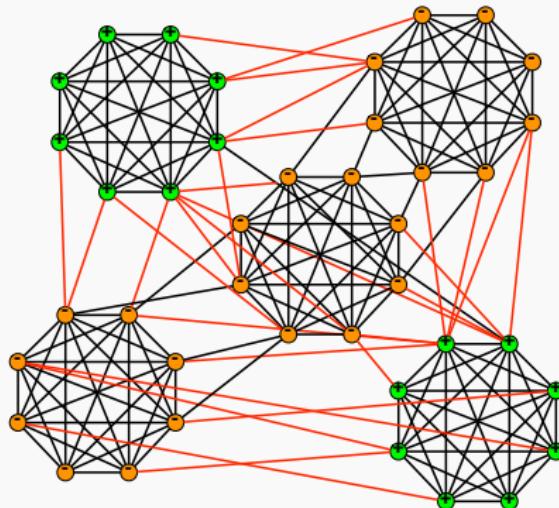
Theorem

The number of mistakes, M , incurred by minimum p -seminorm interpolation with $p := \frac{c}{c-1}$ is bounded by

$$M \leq \begin{cases} N + \frac{4e^2\Phi^2(\mathbf{u})[\log(\Delta_k) + \log(\frac{k}{\Phi(\mathbf{u})}) - 1]}{k^2} & \frac{k\Delta_k}{\Phi(\mathbf{u})} > e^2, \quad p \in (1, 2) \\ N + \frac{4\Phi(\mathbf{u})\Delta_k}{k} & \frac{k\Delta_k}{\Phi(\mathbf{u})} \leq e^2, \quad p = 2 \end{cases}$$

where $c = \max(\log[\frac{k\Delta_k}{\Phi(\mathbf{u})}], 2)$ and $V_1 \cup \dots \cup V_N = V_G$ is any vertex set partition with induced subgraphs G_1, \dots, G_N of maximum wide diameter $\Delta_k := \max\{\Delta_k(G_i) : i = 1, \dots, N\}$, $\Phi(\mathbf{u})$ is the “cutsize” of $\mathbf{u} \in \{-1, 1\}^n$ any consistent labelling.

Example: Prototypical Clusters



$c \times m$ -cliques with ℓ cut edges

- Cover: $N = c$, Energy: $P_p(\mathbf{u}) = \ell \times 2^p$
- Connectivity: $k = m - 1$, Wide Diameter: $\Delta_{m-1} = 2$
- When $p = 2$ then $M \leq c + \frac{8\ell}{m-1}$

Part X

Large Scale Graph-Based Semi-Supervised Learning

SSL in Gigantic Image Collections – 1 [FWT09]

Problem: Even building the graph typically requires quadratic time to compute all pairwise distances – will not scale well to the millions. Something more clever is needed! Sketch follows.

1. Use the completed weighted graph $w_{ij} = \exp(-c\|\mathbf{x}_i - \mathbf{x}_j\|^2)$ as the weighted edges of the Laplacian (note this will never be explicitly calculated)
2. The authors observe that solving $\mathbf{u}^\top L \mathbf{u}$ subject to soft constraints may be approximated by using the few eigenvectors of the Laplacian
3. In the limit of large data rather than finding eigenvectors, instead estimate eigenfunctions given an estimate of the distribution of the data
4. If one further assumes that the density (with a moderate size d)

$$p(\mathbf{s}) = p(s_1)p(s_2)\dots p(s_d)$$

then we can use PCA to find a rotation and estimate each dimension separately

5. Each separate dimension of the density is then estimated by counting elements in the bins of a histograms
6. Computation time will depend only linearly on the number of data points

SSL in Gigantic Image Collections – 2

Data: 79,302,017 images, 445,954 labeled from 386 keywords.



Re-ranking images from 4 keywords in an 80 million image dataset, using 3 labeled pairs for each keyword. Rows from top: "Japanese spaniel", "airbus", "ostrich", "auto". From L to R, the columns show the original image order, results of nearest-neighbors and the results of our eigenfunction approach. By regularizing the solution using eigenfunctions computed from all 80 million images, our semi-supervised scheme outperforms the purely supervised

Part XI

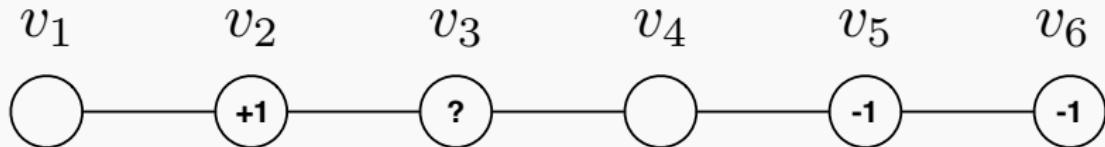
Conclusion

Discussion

- Laplacian-based transduction is versatile
- The graph laplacian may be easily combined with other kernel-based techniques (eg: kernel-Knn, Kernel-PCA)
- There are a large variety of graph “kernels” beyond the Laplacian pseudo-inverse
- Such kernel may be combined in a variety with other kernels of ways allowing the generalization from “transduction” to “induction.”
- Some research issues
 1. Building graphs
 2. Time efficiency – cubic is expensive in practice
 3. Understanding how unlabeled data improves prediction

Problems – 1

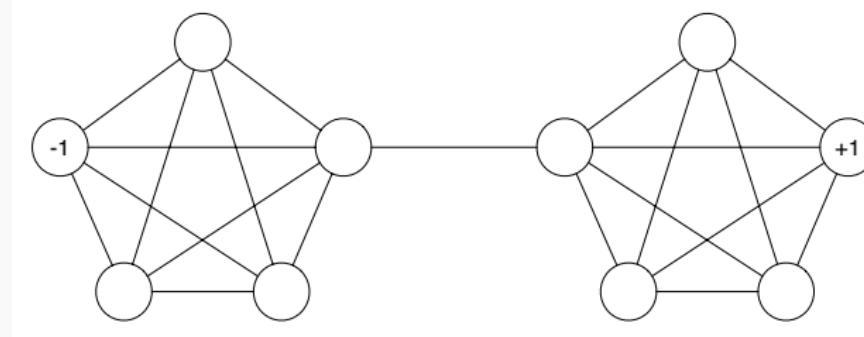
1. Describe, in brief, the differences between semi-supervised learning, supervised learning and unsupervised learning.
2. Given the unlabeled data points $((-2, 0), (-1, -1), (0, 0), (1, 1), (2, 0))$ draw the 2-nearest neighbor graph with respect to the euclidean distance.
3. Give the graph Laplacian for the dataset above.
4. In the labelled graph illustrated below,



- i What is a value of the minimum cut solution at vertex v_3 ?
- ii What is the value of the harmonic energy solution at vertex v_3 ?

Problems – 2

1. Consider the following illustration of the parameterized graph \mathcal{D}_m ($m = 5$ in illustration) that consists of two m -cliques connected by a single bridge edge. The first clique contains a node labelled with a “ -1 ” while the second clique contains a node labelled with a “ $+1$ ”.



2. Consider the labelling of the graph \mathcal{D}_m as resulting from harmonic energy minimisation. Now consider the labelling of \mathcal{D}_m as $m \rightarrow \infty$.
 - i What is the labelling in the limit?
 - ii Provide a justification of your answer to i.
3. [Hard]: Prove the first observation on page 35

Problems – 3

1. [Hard]: Prove the following inequality for the resistance diameter

$$\max_{k \in [n]} L_{kk}^+ \leq \max_{i,j \in [n]} r_{\mathbf{G}}(i,j)$$

2. [Hard]: Prove the equality connecting the effective resistance to the kernel \mathbf{L}^+ on page 55
3. [Hard]: For an unweighted graph prove,

$$\sum_{(i,j) \in \mathcal{G}} r_{\mathbf{G}}(i,j) = n - 1$$

This is trivial for trees. More difficult for generic graphs.

4. [Hard]: In our presentation we only consider equality constraints for interpolation instead of inequality constraints as with an SVM. This problem suggests a justification for this procedure.

Let \mathbf{L} be a graph Laplacian for an n -vertex graph. Consider the kernel (p.d. matrix)

$\mathbf{K} := (\mathbf{L} + \epsilon \mathbf{I})^{-1}$ where \mathbf{I} is the identity matrix and $\epsilon > 0$ is any positive constant.

Prove that for any $\mathbf{x} \in [n]^\ell$ and any $\mathbf{y} \in \{-1, 1\}^\ell$ that

$$\arg \min_{\mathbf{u} \in \mathbb{R}^n : u_{x_1} = y_1, \dots, u_{x_\ell} = y_\ell} \mathbf{u}^\top \mathbf{K}^{-1} \mathbf{u} = \arg \min_{\mathbf{u} \in \mathbb{R}^n : u_{x_1} y_1 \geq 1, \dots, u_{x_\ell} y_\ell \geq 1} \mathbf{u}^\top \mathbf{K}^{-1} \mathbf{u}.$$

Think about the above property in terms of "support vectors" what does this imply for this set of Laplacian-based kernels.

Argue that there exists a positive definite $n \times n$ matrix \mathbf{K} for which this property does not hold.

5. [Technical Hard]: Do the problem on page 42

Suggested Readings

For graph-based semi-supervised learning, please see Chapters 1,2 and 5 from *Introduction to Semi-Supervised Learning*, Xiaojin Zhu and Andrew Goldberg; Morgan and Claypool (2009) [Note: pdf available for download]

For spectral clustering the following tutorial is very good *A Tutorial on Spectral Clustering*, Ulrike Von Luxberg (2007).

The presentation of spectral clustering is closely based on the material in lecture 3 in Michal Valko's course on *Graphs in Machine Learning*. The full lecture notes (slides) has many additional applications of graph-based concepts to machine learning.

References

1. [BC01] A. Blum and S. Chawla *Learning from Labeled and Unlabeled Data Using Graph Mincuts*, 2001
2. [BMN04] M. Belkin, I. Matveeva, and P. Niyogi. *Regularization and semi-supervised learning on large graphs*, 2004
3. [FWT09] B. Fergus, Y. Weiss, and A. Torralba *Semi-supervised Learning in Gigantic Image Collections*, 2009
4. [HL09] M. Herbster and G. Lever. *Predicting the Labelling of a Graph via Minimum p-Seminorm Interpolation*, 2009
5. [L07] U. Luxburg, *A Tutorial on Spectral Clustering*, 2007
6. [ZGL03] X. Zhu, Z. Ghahramani, and J. Lafferty. *Semi-supervised learning using gaussian fields and harmonic functions*, 2003

7. Learning Theory

COMP0078: Supervised Learning

Mark Herbster

22 November 2021

University College London
Department of Computer Science
LT21v3

Acknowledgements

I would like to thank both Shai Shalev-Shwartz and John Shawe-Taylor for permission to use a number of their slides. Please see *Shai Shalev-Shwartz's Lecture Notes (2 & 3) for Understanding Machine Learning 2014* for the original slides.

Plan

- Introduction
- PAC
- Agnostic PAC

Introduction

Part I Introduction

Review and Notation

We recall the supervised learning framework introduced in the first lecture. As convenient we will adopt the notation of *Understanding Machine Learning* [SS14] since this lecture closely follows their presentation. For simplicity we consider the 2-class setting.

[SS14] adopts the following notation for probabilities. If \mathcal{D} is distribution over \mathcal{Z} then if $A \subseteq \mathcal{Z}$ then $\mathcal{D}(A)$ denotes the probability that if z is drawn \mathcal{D} that $z \in A$.

Model : Data is sampled IID from a distribution \mathcal{D} (previously P) over $\mathcal{X} \times \mathcal{Y}$ with $\mathcal{Y} = \{0, 1\}$. The *expected error* (AKA *generalisation error*) of a function $h : \mathcal{X} \rightarrow \mathcal{Y}$ is

$$L_{\mathcal{D}}(h) = \mathcal{D}(\{(x, y) : h(x) \neq y\}) = \mathbb{P}_{(x,y) \sim \mathcal{D}}[h(x) \neq y]$$

in our previous notation $L_{\mathcal{D}}(h) = \mathcal{E}(h) = \int [h(x) \neq y] dP(x, y)$.

The *empirical error* of h with respect to a data set

$S = \{(x_1, y_1), \dots, (x_m, y_m)\}$ is denoted $L_S(h) = \sum_{i=1}^m \frac{1}{m} [h(x_i) \neq y_i]$ (previously $\mathcal{E}_{\text{emp}}(S, h)$).

Validation Set Bound

Theorem

Select an h then for any $\delta \in (0, 1)$. With probability at least $1 - \delta$ over the random sample V of size m from \mathcal{D} we have that

$$L_{\mathcal{D}}(h) \leq L_V(h) + \sqrt{\frac{\ln \frac{1}{\delta}}{2m}}.$$

Interpretation and Application

The generalisation error of a function h may be bounded by its empirical error. Thus in practice we may select predictor h based on a training set S then we may bound its generalisation error by *validating* on a separate set of data V .

Validation Set Bound (Proof 1)

We will use a concentration inequality.

Idea

The law of large numbers states that the empirical average of an IID sequence of random variables $\frac{1}{m} \sum_{i=1}^m Z_i$ will converge to the mean $\mu = E[Z_i]$. Concentration inequalities quantify how the empirical average deviates from the (true) mean when m is finite.

Hoeffding's Inequality

Let Z_1, Z_2, \dots, Z_m be IID Bernoulli random variables where for all i the $\mathbb{P}[Z_i = 1] = p$ and $\forall i : \mathbb{P}[Z_i = 0] = 1 - p$. Let $\bar{Z} := \frac{1}{m} \sum_{i=1}^m Z_i$ then for any $\epsilon > 0$,

$$\mathbb{P}[\bar{Z} > p + \epsilon] \leq \exp(-2m\epsilon^2)$$

$$\mathbb{P}[\bar{Z} < p - \epsilon] \leq \exp(-2m\epsilon^2)$$

Validation Set Bound (Proof 2)

Given h we will bound

$$L_{\mathcal{D}}(h) - L_V(h) = \mathbb{P}_{(x,y) \sim \mathcal{D}}[h(x) \neq y] - \frac{1}{m} \sum_{i=1}^m [h(x_i) \neq y_i]$$

1. Define $Z_i := [h(x_i) \neq y_i]$.
2. Z_1, \dots, Z_m are statistically independent.
3. $\forall i : \mathbb{P}[Z_i] = L_{\mathcal{D}}(h) = \mathbb{P}_{(x,y) \sim \mathcal{D}}[h(x) \neq y]$
4. Applying Hoeffding gives

$$\mathbb{P}[L_{\mathcal{D}}(h) - L_V(h) > \epsilon] \leq \exp(-2\epsilon^2 m)$$

5. Set $\delta := \exp(-2\epsilon^2 m)$ and solving gives the theorem.
6. **Note:** if we use both the upper and lower bounds in Hoeffding then alternately

$$|L_{\mathcal{D}}(h) - L_V(h)| \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2m}}.$$

□

Observations

The above is a nice result with minimal assumptions. However it has drawbacks,

1. The validation bound gives a way to estimate a confidence interval for our generalisation error. However the data V used for validating cannot have been used for training thus it is expensive in data.
2. Often, we have only few examples. Can we choose a model and assess its expected error directly (without splitting the training data)?
3. Rather than only derive confidence bounds about a predictor h we would like instead would like to analyse predictors derived from machine learning algorithms – thus additionally deriving insights into ML algorithms.
4. **Learning theory** studies conditions which ensure the **predictivity** of a learning algorithm:
 - The expected error is close to the empirical error
 - The expected error decreases when the number of samples increases
5. In this lecture we will limit ourselves to the study of **Empirical Risk Minimisation**.

Theories of learning

- Basic approach of Statistical Learning Theory (SLT) is to view learning from a statistical viewpoint.
- Aim of any theory is to model real/ artificial phenomena so that we can better understand/ predict/ exploit them.
- SLT is just one approach to understanding/ predicting/ exploiting learning systems, others include Bayesian inference, inductive inference, statistical physics, traditional statistical analysis.

Theories of learning – cont.

- Each theory makes assumptions about the phenomenon of learning and based on these derives predictions of behaviour as well as algorithms that aim at optimising the predictions.
- Each theory has strengths and weaknesses – the better it captures the details of real world experience, the better the theory and the better the chances of it making accurate predictions and driving good algorithms.

General statistical considerations

- Statistical models (not including Bayesian) begin with an assumption that the data is generated by an underlying distribution \mathcal{D} typically not given explicitly to the learner.
- If we are trying to classify cancerous tissue from healthy tissue, there are two distributions, one for cancerous cells and one for healthy ones.

General statistical considerations cont.

- Usually the distribution subsumes the processes of the natural/artificial world that we are studying.
- Rather than accessing the distribution directly, statistical learning typically assumes that we are given a ‘training sample’ or ‘training set’

$$\mathcal{S} = \{(x_1, y_1), \dots, (x_m, y_m)\}$$

generated identically and independently (i.i.d.) according to the distribution \mathcal{D} .

Generalisation of a learner

- Assume that we have a learning algorithm \mathcal{A} that chooses a hypothesis (function) $\mathcal{A}_{\mathcal{H}}(\mathcal{S})$ from a hypothesis (function) space \mathcal{H} in response to the training set \mathcal{S} .
- We are interested in the study of $L_{\mathcal{D}}(\mathcal{A}_{\mathcal{H}}(\mathcal{S}))$ i.e., the generalisation error of our algorithm $\mathcal{A}_{\mathcal{H}}(\mathcal{S})$ which is a random variable dependent on the draw of \mathcal{S} .
- In particular we will study empirical risk minimization

$$\text{ERM}_{\mathcal{H}}(\mathcal{S}) := \arg \min_{h \in \mathcal{H}} L_{\mathcal{S}}(h)$$

- **Note:** There may be many possible empirical risk minimisers we assume $\text{ERM}_{\mathcal{H}}(\mathcal{S})$ outputs an arbitrary one.

Example of Generalisation I

- We consider the Breast Cancer dataset from the UCI repository.
- Use the simple Parzen window classifier described by Bernhard Schölkopf: weight vector is

$$\mathbf{w}^+ - \mathbf{w}^-$$

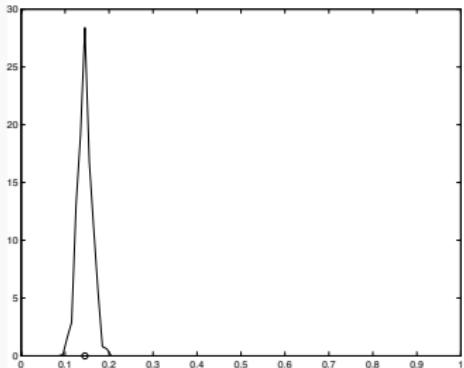
where \mathbf{w}^+ is the average of the positive training examples and \mathbf{w}^- is average of negative training examples. Threshold is set so hyperplane bisects the line joining these two points.

Example of Generalisation II

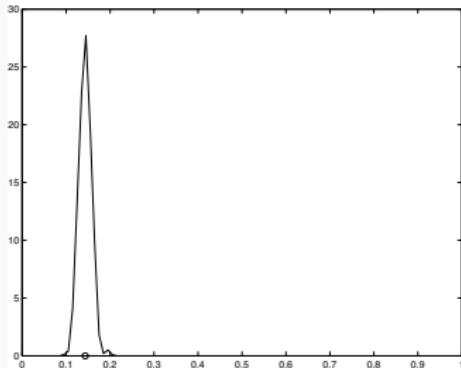
- Given a size m of the training set, by repeatedly drawing random training sets S we estimate the distribution of $L_D(A_{\mathcal{H}}^{\text{parzen}}(S))$ by using the test set error as a proxy for the true generalisation.
- We plot the histogram and the average of the distribution for various sizes of training set (342, 273, 205, 137, 68, 34, 27, 20, 14, 7)

Error distributions – 1

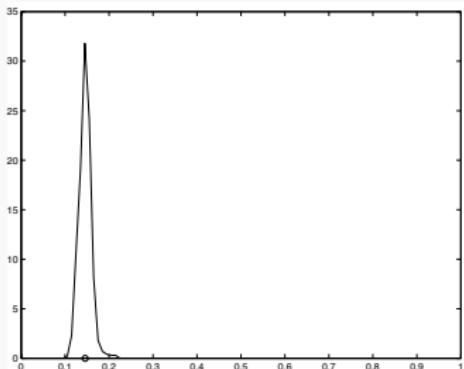
dataset size: 342



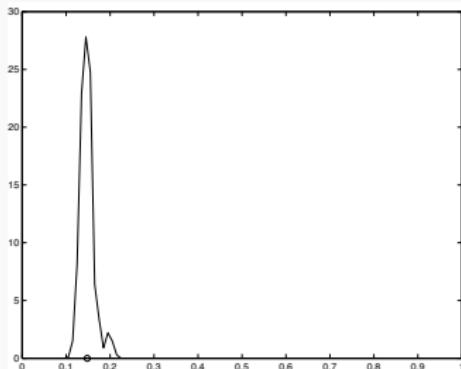
dataset size: 273



dataset size: 205

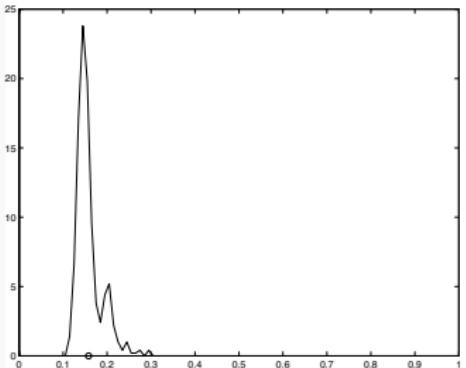


dataset size: 137

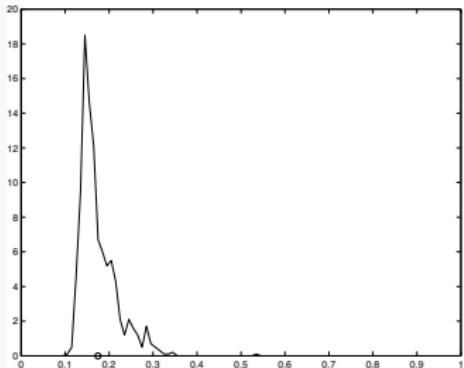


Error distributions – 2

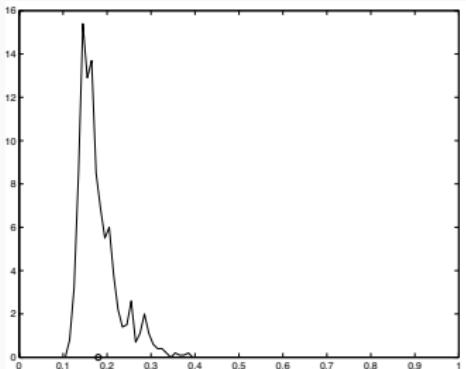
dataset size: 68



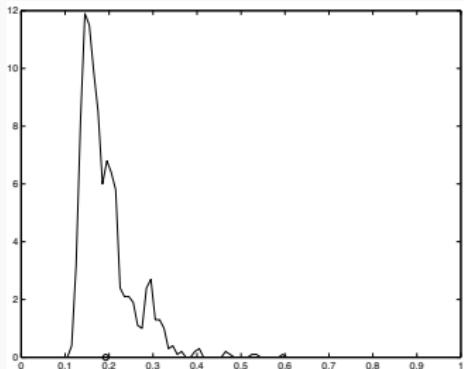
dataset size: 34



dataset size: 27

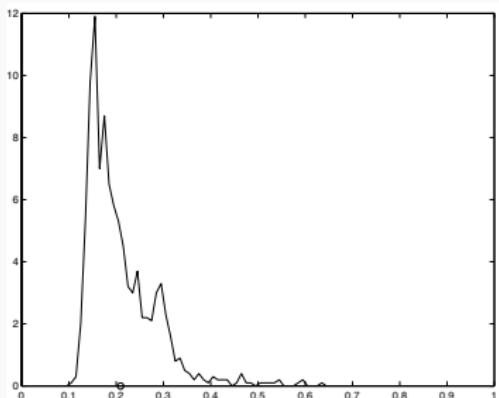


dataset size: 20

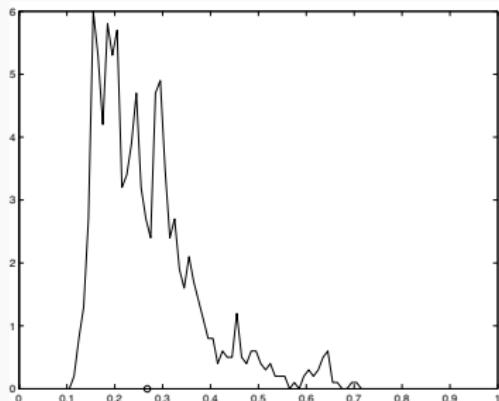


Error distributions – 3

dataset size: 14



dataset size: 7



Bayes risk and consistency

- Traditional statistics has concentrated on analysing

$$\lim_{m \rightarrow \infty} E_{S_m}[L_{\mathcal{D}}(\mathcal{A}(S_m))].$$

where S_m denotes a training set of size m .

- For example consistency of a classification algorithm is the function with the lowest possible risk, often referred to as the Bayes risk.
- Recall our discussion of kNN .

Expected versus confident bounds

- For a finite sample the generalisation $L_{\mathcal{D}}(\mathcal{A}(S_m))$ has a distribution depending on the algorithm, function class and sample size m .
- Traditional statistics as indicated above has concentrated on the mean of this distribution – but this quantity can be misleading, eg for low fold cross-validation.

Expected versus confident bounds (cont.)

- Statistical learning theory has preferred to analyse the tail of the distribution, finding a bound which holds with high probability.
- This looks like a statistical test – significant at a 1% confidence means that the chances of the conclusion not being true are less than 1% over random samples of that size.
- Observe in the previous example on the breast cancer data the mean generalisation error only shifted moderately compared to the movement of the “tail” of the distribution.
- This is also the source of the acronym PAC: probably approximately correct, the ‘confidence’ parameter δ is the probability that we have been misled by the training set.

Part II

PAC Learning

Realisability Assumption

Realisability Assumption: (We will later remove this assumption)

- We will assume that there exists some true function f^* so that for all $x \in \mathcal{X}$ we have $f^*(x) = y$ (i.e., there exists a classifier with zero error).
- An implication of this assumption is that we can treat \mathcal{D} now as only a distribution over \mathcal{X} (as opposed to $\mathcal{X} \times \mathcal{Y}$).
- To indicate the dependence on f^* we will now use the notation

$$L_{\mathcal{D}, f^*}(h) = \mathbb{P}_{x \sim \mathcal{D}}[h(x) \neq f^*(x)]$$

- Can we find an algorithm \mathcal{A} so that $h = \mathcal{A}(S)$ s.t. $L_{\mathcal{D}, f^*}(h)$ is small?

Can only be Approximately correct

- **Claim:** We can't hope to find h s.t. $L_{(\mathcal{D}, f^*)}(h) = 0$
- Proof: for every $\epsilon \in (0, 1)$ take $\mathcal{X} = \{x_1, x_2\}$ and $\mathcal{D}(\{x_1\}) = 1 - \epsilon$, $\mathcal{D}(\{x_2\}) = \epsilon$
- The probability not to see x_2 at all among m i.i.d. examples is $(1 - \epsilon)^m \approx e^{-\epsilon m}$
- So, if $\epsilon \ll 1/m$ we're likely not to see x_2 at all, but then we can't know its label
- **Relaxation:** We'd be happy with $L_{(\mathcal{D}, f^*)}(h) \leq \epsilon$, where ϵ is user-specified

Can only be Probably correct

- Recall that the input to the learner is randomly generated
- There's always a (very small) chance to see the same example again and again
- **Claim:** No algorithm can guarantee $L_{(\mathcal{D}, f^*)}(h) \leq \epsilon$ for sure
- **Relaxation:** We'd allow the algorithm to fail with probability δ , where $\delta \in (0, 1)$ is user-specified
Here, the probability is over the random choice of examples

Probably Approximately Correct (PAC) learning

- The learner doesn't know \mathcal{D} and f^* .
- The learner receives accuracy parameter ϵ and confidence parameter δ
- The learner can ask for training data, S , containing $m(\epsilon, \delta)$ examples (that is, the number of examples can depend on the value of ϵ and δ , but it can't depend on \mathcal{D} or f^*)
- Learner should output a hypothesis h s.t. with probability of at least $1 - \delta$ it holds that $L_{\mathcal{D}, f^*}(h) \leq \epsilon$
- That is, the learner should be **P**robably (with probability at least $1 - \delta$) **A**pproximately (up to accuracy ϵ) **C**orrect

No Free Lunch and Prior Knowledge

No Free Lunch

- Suppose that $|\mathcal{X}| = \infty$
- For any finite $C \subset \mathcal{X}$ take \mathcal{D} to be uniform distribution over C
- If number of training examples is $m \leq |C|/2$ the learner has no knowledge on at least half the elements in C
- Formalizing the above, it can be shown that:

Theorem (No Free Lunch)

Fix $\delta \in (0, 1)$, $\epsilon < 1/2$. For every learner A and training set size m , there exists \mathcal{D}, f^* such that with probability of at least δ over the generation of a training data, S , of m examples, it holds that

$$L_{\mathcal{D}, f^*}(A(S)) \geq \epsilon.$$

Remark: $L_{\mathcal{D}, f^*}(\text{random guess}) = 1/2$, so the theorem states that you can't be better than a random guess. **Note:** This is stronger result in expectation than given in lecture 1. See Problem 5.3 in [SS14].

Prior Knowledge

- Give more knowledge to the learner: the target f comes from some **hypothesis class**, $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$
- The learner knows \mathcal{H}
- Is it possible to PAC learn now ?
- Of course, the answer depends on \mathcal{H} since the No Free Lunch theorem tells us that for $\mathcal{H} = \mathcal{Y}^{\mathcal{X}}$ the problem is not solvable ...

PAC Learning of Finite Hypothesis Classes

Learning Finite Classes

- Assume that \mathcal{H} is a finite hypothesis class
 - E.g.: \mathcal{H} is all the functions from \mathcal{X} to \mathcal{Y} that can be implemented using a Python program of length at most b
- Use the **Consistent** learning rule:
 - Input: \mathcal{H} and $S = (x_1, y_1), \dots, (x_m, y_m)$
 - Output: any $h \in \mathcal{H}$ s.t. $\forall i, y_i = h(x_i)$
- This is also called **Empirical Risk Minimization (ERM)**

$\text{ERM}_{\mathcal{H}}(S)$

- Input: training set $S = (x_1, y_1), \dots, (x_m, y_m)$
- Define the empirical risk: $L_S(h) = \frac{1}{m} |\{i : h(x_i) \neq y_i\}|$
- Output: any $h \in \mathcal{H}$ that minimizes $L_S(h)$

Learning Finite Classes

Theorem

Fix $\epsilon, \delta \in (0, 1)$. If

$$m \geq \frac{\log(|\mathcal{H}|/\delta)}{\epsilon}$$

then for every \mathcal{D}, f^* , with probability of at least $1 - \delta$ (with respect to the randomly sampled training set S of size m),

$$L_{\mathcal{D}, f^*}(\text{ERM}_{\mathcal{H}}(S)) \leq \epsilon.$$

Interpretation

By rearranging we have,

$$L_{\mathcal{D}, f^*}(\text{ERM}_{\mathcal{H}}(S)) \leq \frac{\log |\mathcal{H}| + \log \frac{1}{\delta}}{m}.$$

Thus **ERM** in the realisable case generalisation error decreases **linearly** in the number of samples m and increases **logarithmically** in the size of the hypothesis class.

Proof

- Let $S|_x = (x_1, \dots, x_m)$ be the instances of the training set
- We would like to prove:

$$\mathcal{D}^m(\{S|_x : L_{(\mathcal{D}, f^*)}(\text{ERM}_{\mathcal{H}}(S)) > \epsilon\}) \leq \delta$$

- Let \mathcal{H}_B be the set of “bad” hypotheses,

$$\mathcal{H}_B = \{h \in \mathcal{H} : L_{(\mathcal{D}, f^*)}(h) > \epsilon\}$$

- Let M be the set of “misleading” samples,

$$M = \{S|_x : \exists h \in \mathcal{H}_B, L_S(h) = 0\}$$

- Observe:

$$\{S|_x : L_{(\mathcal{D}, f^*)}(\text{ERM}_{\mathcal{H}}(S)) > \epsilon\} \subseteq M = \bigcup_{h \in \mathcal{H}_B} \{S|_x : L_S(h) = 0\}$$

Proof (Cont.)

Lemma (Union bound)

For any two sets A, B and a distribution \mathcal{D} we have

$$\mathcal{D}(A \cup B) \leq \mathcal{D}(A) + \mathcal{D}(B).$$

- We have shown:

$$\{S|_x : L_{(\mathcal{D}, f^*)}(\text{ERM}_{\mathcal{H}}(S)) > \epsilon\} \subseteq \bigcup_{h \in \mathcal{H}_B} \{S|_x : L_S(h) = 0\}$$

- Therefore, using the union bound

$$\begin{aligned} \mathcal{D}^m(\{S|_x : L_{(\mathcal{D}, f^*)}(\text{ERM}_{\mathcal{H}}(S)) > \epsilon\}) \\ \leq \sum_{h \in \mathcal{H}_B} \mathcal{D}^m(\{S|_x : L_S(h) = 0\}) \\ \leq |\mathcal{H}_B| \max_{h \in \mathcal{H}_B} \mathcal{D}^m(\{S|_x : L_S(h) = 0\}) \end{aligned}$$

Proof (Cont.)

- Observe:

$$\mathcal{D}^m(\{S|_x : L_S(h) = 0\}) = (1 - L_{\mathcal{D}, f^*}(h))^m$$

- If $h \in \mathcal{H}_B$ then $L_{\mathcal{D}, f^*}(h) > \epsilon$ and therefore

$$\mathcal{D}^m(\{S|_x : L_S(h) = 0\}) < (1 - \epsilon)^m$$

- We have shown:

$$\mathcal{D}^m(\{S|_x : L_{(\mathcal{D}, f^*)}(\text{ERM}_{\mathcal{H}}(S)) > \epsilon\}) < |\mathcal{H}_B| (1 - \epsilon)^m$$

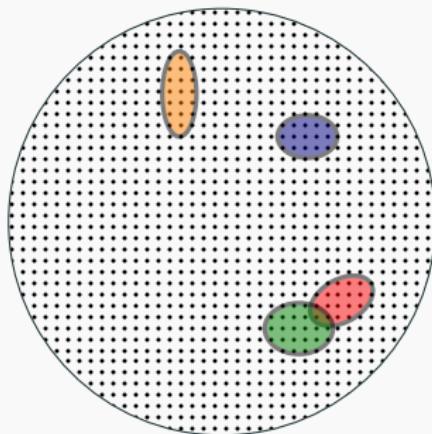
- Finally, using $1 - \epsilon \leq e^{-\epsilon}$ and $|\mathcal{H}_B| \leq |\mathcal{H}|$ we conclude:

$$\mathcal{D}^m(\{S|_x : L_{(\mathcal{D}, f^*)}(\text{ERM}_{\mathcal{H}}(S)) > \epsilon\}) < |\mathcal{H}| e^{-\epsilon m}$$

- The right-hand side would be $\leq \delta$ if $m \geq \frac{\log(|\mathcal{H}|/\delta)}{\epsilon}$.

□

Illustrating the use of the union bound



- Each point is a possible sample $S|x$. Each colored oval represents misleading samples for some $h \in \mathcal{H}_B$. The probability mass of each such oval is at most $(1 - \epsilon)^m$. But, the algorithm might err if it samples $S|x$ from any of these ovals.

The Fundamental Theorem of Learning Theory

PAC learning

Definition (PAC learnability)

A hypothesis class \mathcal{H} is PAC learnable if there exists a function $m_{\mathcal{H}} : (0, 1)^2 \rightarrow \mathbb{N}$ and a learning algorithm with the following property:

- for every $\epsilon, \delta \in (0, 1)$
- for every distribution \mathcal{D} over \mathcal{X} , and for every labeling function $f^* : \mathcal{X} \rightarrow \{0, 1\}$

when running the learning algorithm on $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. examples generated by \mathcal{D} and labeled by f^* , the algorithm returns a hypothesis h such that, with probability of at least $1 - \delta$ (over the choice of the examples), $L_{(\mathcal{D}, f^*)}(h) \leq \epsilon$.

$m_{\mathcal{H}}$ is called the **sample complexity** of learning \mathcal{H}

PAC learning

Leslie Valiant, Turing award 2010

For transformative contributions to the theory of computation, including the theory of probably approximately correct (PAC) learning, the complexity of enumeration and of algebraic computation, and the theory of parallel and distributed computing.



What is learnable and how to learn?

- We have shown:

Corollary

Let \mathcal{H} be a finite hypothesis class.

- \mathcal{H} is PAC learnable with sample complexity $m_{\mathcal{H}}(\epsilon, \delta) \leq \frac{\log(|\mathcal{H}|/\delta)}{\epsilon}$
 - This sample complexity is obtained by using the $\text{ERM}_{\mathcal{H}}$ learning rule
-
- What about infinite hypothesis classes?
 - What is the sample complexity of a given class?
 - Is there a generic learning algorithm that achieves the optimal sample complexity ?

What is learnable and how to learn?

The fundamental theorem of statistical learning:

- The sample complexity is characterized by the **VC dimension**
- The ERM learning rule is a generic (near) optimal learner

Chervonenkis



Vapnik

The VC dimension — Motivation

- Suppose we got a training set $S = (x_1, y_1), \dots, (x_m, y_m)$
- We try to explain the labels using a hypothesis from \mathcal{H}
- Then, ooops, the labels we received are incorrect and we get the same instances with different labels, $S' = (x_1, y'_1), \dots, (x_m, y'_m)$
- We again try to explain the labels using a hypothesis from \mathcal{H}
- If this works for us, no matter what the labels are, then something is fishy ...
- Formally, if \mathcal{H} allows all functions over some set C of size m , then based on the No Free Lunch, we can't learn from, say, $m/2$ examples

The VC dimension — Formal Definition

- Let $C = \{x_1, \dots, x_{|C|}\} \subset \mathcal{X}$
- Let \mathcal{H}_C be the restriction of \mathcal{H} to C , namely, $\mathcal{H}_C = \{h_C : h \in \mathcal{H}\}$ where $h_C : C \rightarrow \{-1, 1\}$ is s.t. $h_C(x_i) = h(x_i)$ for every $x_i \in C$
- Observe: we can represent each h_C as the vector $(h(x_1), \dots, h(x_{|C|})) \in \{\pm 1\}^{|C|}$
- Therefore: $|\mathcal{H}_C| \leq 2^{|C|}$
- We say that \mathcal{H} **shatters** C if $|\mathcal{H}_C| = 2^{|C|}$
- $\text{VCdim}(\mathcal{H}) = \sup\{|C| : \mathcal{H} \text{ shatters } C\}$
- That is, the VC dimension is the maximal size of a set C such that \mathcal{H} gives no prior knowledge w.r.t. C

VC-Dimension Illustrated

- We may visualise a hypothesis space \mathcal{H} as a **sign matrix** (potentially infinite). The rows representing **hypotheses** and the columns **instances**. The VC-dimension $\text{VCdim}(\mathcal{H})$ is determined by the maximal $2^{\text{VCdim}(\mathcal{H})} \times \text{VCdim}(\mathcal{H})$ submatrix in which each row is distinct, i.e, all $2^{\text{VCdim}(\mathcal{H})}$ possible sign patterns of length $\text{VCdim}(\mathcal{H})$ are present.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
h_1	+	+	-	+	-	+	-	-	+
h_2	-	-	+	+	+	+	-	+	+
h_3	-	-	-	-	+	-	-	-	-
h_4	+	+	-	-	+	+	-	+	-
h_5	-	-	-	+	-	+	-	+	+
h_6	-	-	+	+	-	+	-	-	-
h_7	+	-	+	-	-	+	+	+	+
h_8	-	-	+	+	+	+	-	-	-

$$\text{VCdim}(\mathcal{H}) = 3$$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
h_1	+	+	-	+	-	+	-	-	+
h_2	-	-	+	+	+	+	-	+	+
h_3	-	-	+	-	+	-	-	-	-
h_4	+	+	-	-	+	+	-	+	-
h_5	-	-	-	+	-	+	-	+	+
h_6	-	-	+	+	-	+	-	-	-
h_7	+	-	+	-	-	+	+	+	+
h_8	-	-	+	+	+	+	-	-	-

$$\text{VCdim}(\mathcal{H}) = 2$$

VC dimension — Examples

To show that $\text{VCdim}(\mathcal{H}) = d$ we need to show that:

1. There exists a set C of size d which is shattered by \mathcal{H} .
2. Every set C of size $d + 1$ is not shattered by \mathcal{H} .

VC dimension — Examples

Threshold functions: $\mathcal{X} = \mathbb{R}$, $\mathcal{H} = \{x \mapsto \text{sign}(x - \theta) : \theta \in \mathbb{R}\}$

- Show that $\{0\}$ is shattered
- Show that any two points cannot be shattered

Assume for this lecture that

$$\text{sign}(z) := \begin{cases} 1 & z \geq 0 \\ -1 & z < 0 \end{cases}$$

This differ from the usual definition where $\text{sign}(0) = 0$.

VC dimension — Examples

Intervals: $\mathcal{X} = \mathbb{R}$, $\mathcal{H} = \{h_{a,b} : a < b \in \mathbb{R}\}$, where $h_{a,b}(x) = 1$ iff $x \in [a, b]$

- Show that $\{0, 1\}$ is shattered
- Show that any three points cannot be shattered

VC dimension — Examples

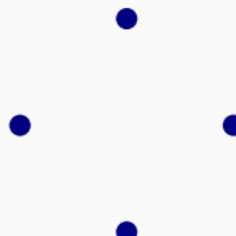
Axis aligned rectangles: $\mathcal{X} = \mathbb{R}^2$,

$\mathcal{H} = \{h_{(a_1, a_2, b_1, b_2)} : a_1 < a_2 \text{ and } b_1 < b_2\}$, where

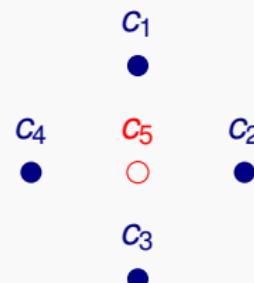
$h_{(a_1, a_2, b_1, b_2)}(x_1, x_2) = 1 \text{ iff } x_1 \in [a_1, a_2] \text{ and } x_2 \in [b_1, b_2]$

Show:

Shattered



Not Shattered



VC dimension — Examples

Finite classes:

- Show that the VC dimension of a finite \mathcal{H} is at most $\log_2(|\mathcal{H}|)$.
- Show that there can be arbitrary gap between $\text{VCdim}(\mathcal{H})$ and $\log_2(|\mathcal{H}|)$

VC dimension — Examples

Halfspaces: $\mathcal{X} = \mathbb{R}^n$, $\mathcal{H} = \{\mathbf{x} \mapsto \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle) : \mathbf{w} \in \mathbb{R}^n\}$

- Show that $\{\eta_1, \dots, \eta_n\}$ is shattered
- Show that any $n + 1$ points cannot be shattered

VC dimension (Large Margin Halfspaces)

Define the inner product space of bounded square summable sequences $\ell_2 := \{\mathbf{x} \in \mathbb{R}^\infty : \sum_{i=1}^{\infty} x_i^2 < \infty\}$ with inner product $\langle \mathbf{x}, \mathbf{x}' \rangle = \sum_{i=1}^{\infty} x_i x'_i$.

Definition

Given an $X \subset \ell_2$ and a $\Lambda \in (0, \infty)$ then define,

$$\mathcal{H}_{X,\Lambda} = \{\mathbf{x} \mapsto \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle) : \mathbf{x} \in X, \mathbf{w} \in \ell_2, \|\mathbf{w}\| \leq \Lambda, |\langle \mathbf{w}, \mathbf{x} \rangle| \geq 1\}$$

- Observe that $1/\|\mathbf{w}\|$ is the margin.

Theorem

$$\text{VCdim}(\mathcal{H}_{X,\Lambda}) \leq \Lambda^2 \max_{\mathbf{x} \in X} \|\mathbf{x}\|^2$$

- Prove theorem (Hint : think “online learning”)
- Show that for every $\Lambda \in (0, \infty)$ there exists an X such that $\text{VCdim}(\mathcal{H}_{X,\Lambda}) = \lfloor \Lambda^2 \max_{\mathbf{x} \in X} \|\mathbf{x}\|^2 \rfloor$.

The Fundamental Theorem of Statistical Learning

Theorem (The Fundamental Theorem of Statistical Learning)

Let \mathcal{H} be a hypothesis class of binary classifiers. Then, there are absolute constants C_1, C_2 such that the sample complexity of PAC learning \mathcal{H} is

$$C_1 \frac{\text{VCdim}(\mathcal{H}) + \log(1/\delta)}{\epsilon} \leq m_{\mathcal{H}}(\epsilon, \delta) \leq C_2 \frac{\text{VCdim}(\mathcal{H}) \log(1/\epsilon) + \log(1/\delta)}{\epsilon}$$

Furthermore, this sample complexity is achieved by the ERM learning rule.

Proof of the lower bound – main ideas

- Suppose $\text{VCdim}(\mathcal{H}) = d$ and let $C = \{x_1, \dots, x_d\}$ be a shattered set
- Consider the distribution \mathcal{D} supported on C s.t.

$$\mathcal{D}(\{x_i\}) = \begin{cases} 1 - 4\epsilon & \text{if } i = 1 \\ 4\epsilon/(d-1) & \text{if } i > 1 \end{cases}$$

- If we see m i.i.d. examples then the expected number of examples from $C \setminus \{x_1\}$ is $4\epsilon m$
- If $m < \frac{d-1}{8\epsilon}$ then $4\epsilon m < \frac{d-1}{2}$ and therefore, we have no information on the labels of at least half the examples in $C \setminus \{x_1\}$
- Best we can do is to guess, but then our error is $\geq \frac{1}{2} \cdot 2\epsilon = \epsilon$

Proof of the upper bound – main ideas

- Recall our proof for finite class:
 - For a single hypothesis, we've shown that the probability of the event: $L_S(h) = 0$ given that $L_{(\mathcal{D}, f^*)}(h) > \epsilon$ is at most $e^{-\epsilon m}$
 - Then we applied the union bound over all “bad” hypotheses, to obtain the bound on ERM failure: $|\mathcal{H}| e^{-\epsilon m}$
- If \mathcal{H} is infinite, or very large, the union bound yields a meaningless bound

Proof of the upper bound – main ideas

- The two samples trick: show that

$$\begin{aligned} & \mathbb{P}_{S \sim \mathcal{D}^m} [\exists h \in \mathcal{H}_B : L_S(h) = 0] \\ & \leq 2\mathbb{P}_{S, T \sim \mathcal{D}^m} [\exists h \in \mathcal{H}_B : L_S(h) = 0 \text{ and } L_T(h) \geq \epsilon/2] \end{aligned}$$

- Symmetrization: Since S, T are i.i.d., we can think on first sampling $2m$ examples and then splitting them to S, T at random
- If we fix h , and $S \cup T$, the probability to have $L_S(h) = 0$ while $L_T(h) \geq \epsilon/2$ is $\leq e^{-\epsilon m/4}$
- Once we fixed $S \cup T$, we can take a union bound only over $\mathcal{H}_{S \cup T}$ (i.e., the restriction of \mathcal{H} to “columns” denoted by $S \cup T$) !

Proof of the upper bound – main ideas

- The growth function $\Pi_{\mathcal{H}}(m)$ is the maximum cardinality of the set of functions in \mathcal{H} when restricted to m points ("columns"). Thus,

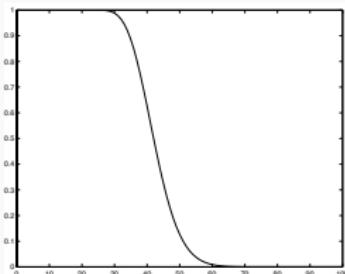
$$\Pi_{\mathcal{H}}(m) := \max_{C \subset \mathcal{X}} \{|\mathcal{H}_C| : |C| = m\}.$$

- Thus $\text{VCdim}(\mathcal{H}) = \max\{m : \Pi_{\mathcal{H}}(m) = 2^m\}$

Lemma: Sauer-Shelah-Perles²-Vapnik-Chervonenkis

Let \mathcal{H} be a hypothesis class with $d = \text{VCdim}(\mathcal{H}) < \infty$ then the growth function may be bounded as,

$$\Pi_{\mathcal{H}}(m) = \begin{cases} 2^m & m \leq d \\ \leq \sum_{i=0}^d \binom{m}{i} \leq \left(\frac{em}{d}\right)^d & m > d \end{cases}$$



$\Pi_{\mathcal{H}}(m)/2^m$ for linear functions in a 20 dimensional space.

- The VC-dimension is the point where the curve rapidly decreases. I.e, where we now have a polynomial growth in the number of distinct hypotheses as opposed to exponential.
- It is precisely this slowed growth that allows the bound to be proven.

Agnostic Pac

Part III Agnostic Pac

Relaxing the realizability assumption – Agnostic PAC learning

- So far we assumed that labels are generated by some $f^* \in \mathcal{H}$
- This assumption may be too strong
- Relax the realizability assumption by replacing the “target labeling function” with a more flexible notion, a data-labels generating distribution
- We now return to the assumption that \mathcal{D} be a distribution over $\mathcal{X} \times \mathcal{Y}$.
- We now use $L_{\mathcal{D}}(h)$ (as opposed to $L_{(\mathcal{D}, f^*)}(h)$) :

$$L_{\mathcal{D}}(h) \stackrel{\text{def}}{=} \mathbb{P}_{(x,y) \sim \mathcal{D}}[h(x) \neq y] \stackrel{\text{def}}{=} \mathcal{D}(\{(x, y) : h(x) \neq y\})$$

- We redefine the “approximately correct” notion to

$$L_{\mathcal{D}}(A(S)) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \epsilon$$

- **Note:** We will only prove the result for finite \mathcal{H} however, the proof for instead hypotheses classes with finite $\text{VCdim}(\mathcal{H})$ carries over in the same way in the agnostic setting.

The General PAC Learning Problem

We wish to Probably Approximately Solve:

$$\min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) \quad \text{where} \quad L_{\mathcal{D}}(h) = \mathbb{P}_{(x,y) \sim \mathcal{D}}[h(x) \neq y].$$

- Learner knows \mathcal{H}
- Learner receives accuracy parameter ϵ and confidence parameter δ
- Learner can decide on training set size m based on ϵ, δ
- Learner doesn't know \mathcal{D} but can sample $S \sim \mathcal{D}^m$
- Using S the learner outputs some hypothesis $\mathcal{A}(S)$
- We want that with probability of at least $1 - \delta$ over the choice of S , the following would hold: $L_{\mathcal{D}}(\mathcal{A}(S)) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \epsilon$

Formal definition

A hypothesis class \mathcal{H} is agnostic PAC learnable if there exists a function $m_{\mathcal{H}} : (0, 1)^2 \rightarrow \mathbb{N}$ and a learning algorithm, \mathcal{A} , with the following property: for every $\epsilon, \delta \in (0, 1)$, $m \geq m_{\mathcal{H}}(\epsilon, \delta)$, and distribution \mathcal{D} over $\mathcal{X} \times \mathcal{Y}$,

$$\mathcal{D}^m \left(\left\{ S \in (\mathcal{X} \times \mathcal{Y})^m : L_{\mathcal{D}}(\mathcal{A}(S)) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \epsilon \right\} \right) \geq 1 - \delta$$

Learning via Uniform Convergence

Representative Sample

Definition (ϵ -representative sample)

A training set S is called ϵ -representative if

$$\forall h \in \mathcal{H}, \quad |L_S(h) - L_{\mathcal{D}}(h)| \leq \epsilon .$$

Representative Sample

Lemma

Assume that a training set S is $\frac{\epsilon}{2}$ -representative. Then, any output of $\text{ERM}_{\mathcal{H}}(S)$, namely any $h_S \in \arg \min_{h \in \mathcal{H}} L_S(h)$, satisfies

$$L_{\mathcal{D}}(h_S) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \epsilon.$$

Proof: For every $h \in \mathcal{H}$,

$$L_{\mathcal{D}}(h_S) \leq L_S(h_S) + \frac{\epsilon}{2} \leq L_S(h) + \frac{\epsilon}{2} \leq L_{\mathcal{D}}(h) + \frac{\epsilon}{2} + \frac{\epsilon}{2} = L_{\mathcal{D}}(h) + \epsilon$$

Uniform Convergence is Sufficient for Learnability

Definition (uniform convergence)

\mathcal{H} has the *uniform convergence property* if there exists a function $m_{\mathcal{H}}^{\text{UC}} : (0, 1)^2 \rightarrow \mathbb{N}$ such that for every $\epsilon, \delta \in (0, 1)$, and every distribution \mathcal{D} ,

$$\mathcal{D}^m(\{S \in \mathcal{Z}^m : S \text{ is } \epsilon\text{-representative}\}) \geq 1 - \delta$$

with $m \geq m_{\mathcal{H}}^{\text{UC}}(\epsilon, \delta)$.

Corollary

- If \mathcal{H} has the uniform convergence property with a function $m_{\mathcal{H}}^{\text{UC}}$ then \mathcal{H} is agnostically PAC learnable with the sample complexity $m_{\mathcal{H}}(\epsilon, \delta) \leq m_{\mathcal{H}}^{\text{UC}}(\epsilon/2, \delta)$.
- Furthermore, in that case, the $\text{ERM}_{\mathcal{H}}$ paradigm is a successful agnostic PAC learner for \mathcal{H} .

Finite Classes are Agnostic PAC Learnable

We will prove the following:

Theorem

Assume \mathcal{H} is finite. Then, \mathcal{H} is agnostically PAC learnable using the $\text{ERM}_{\mathcal{H}}$ algorithm with sample complexity

$$m_{\mathcal{H}}(\epsilon, \delta) \leq \left\lceil \frac{2 \log(2|\mathcal{H}|/\delta)}{\epsilon^2} \right\rceil.$$

Interpretation (compare to realisable case see page 31)

By rearranging we have,

$$L_{\mathcal{D}}(\text{ERM}_{\mathcal{H}}(S)) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + 2 \sqrt{\frac{\log |\mathcal{H}| + \log \frac{2}{\delta}}{2m}}.$$

Comparing to the realisable case generalisation error now decreases \sqrt{m} rate as opposed to linearly.

Proof (cont.)

Proof: It suffices to show that \mathcal{H} has the uniform convergence property with

$$m_{\mathcal{H}}^{\text{uc}}(\epsilon, \delta) \leq \left\lceil \frac{\log(2|\mathcal{H}|/\delta)}{2\epsilon^2} \right\rceil.$$

- To show uniform convergence, we need:

$$\mathcal{D}^m(\{S : \exists h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}) < \delta.$$

- Using the union bound (compare to Page 33):

$$\begin{aligned}\mathcal{D}^m(\{S : \exists h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}) &= \\ \mathcal{D}^m(\cup_{h \in \mathcal{H}} \{S : |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}) &\leq \\ \sum_{h \in \mathcal{H}} \mathcal{D}^m(\{S : |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}).\end{aligned}$$

Proof (cont.)

This is the same argument we used on Page 8 (Validation Set Bound Proof)

- Recall: $L_{\mathcal{D}}(h) = \mathbb{P}_{(x,y) \sim \mathcal{D}}[h(x) \neq y]$ and $L_S(h) = \frac{1}{m} \sum_{i=1}^m [h(x_i) \neq y_i]$.
- Define $Z_i := [h(x_i) \neq y_i]$.
- Z_1, \dots, Z_m are statistically independent.

Recall:

Hoeffding's Inequality

Let Z_1, Z_2, \dots, Z_m be IID Bernoulli random variables where for all i the $\mathbb{P}[Z_i = 1] = p$ and $\forall i : \mathbb{P}[Z_i = 0] = 1 - p$. Let $\bar{Z} := \frac{1}{m} \sum_{i=1}^m Z_i$ then for any $\epsilon > 0$,

$$\mathbb{P}[\bar{Z} > p + \epsilon] \leq \exp(-2m\epsilon^2)$$

$$\mathbb{P}[\bar{Z} < p - \epsilon] \leq \exp(-2m\epsilon^2)$$

This implies:

$$\mathcal{D}^m(\{S : |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}) \leq 2 \exp(-2m\epsilon^2).$$

Proof (cont.)

We have shown:

$$\mathcal{D}^m(\{S : \exists h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}) \leq 2|\mathcal{H}| \exp(-2m\epsilon^2)$$

So, if $m \geq \frac{\log(2|\mathcal{H}|/\delta)}{2\epsilon^2}$ then the right-hand side is at most δ as required.

□

Error Decomposition

- Let $h_S = \text{ERM}_{\mathcal{H}}(S)$. We can decompose the risk of h_S as:

$$L_{\mathcal{D}}(h_S) = \epsilon_{\text{app}} + \epsilon_{\text{est}}$$



- The approximation error,** $\epsilon_{\text{app}} = \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$:
 - How much risk do we have due to restricting to \mathcal{H}
 - Doesn't depend on S
 - Decreases with the complexity (size, or VC dimension) of \mathcal{H}
- The estimation error,** $\epsilon_{\text{est}} = L_{\mathcal{D}}(h_S) - \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$:
 - Result of L_S being only an estimate of $L_{\mathcal{D}}$
 - Decreases with the size of S
 - Increases with the complexity of \mathcal{H}
- Bias/Complexity :** Choosing $\mathcal{H}' \supset \mathcal{H}$ leads to decreased ϵ_{app} while ϵ_{est} is increased.

Summary

- PAC-learning provides guarantees on the generalisation error under worst-case distributional assumptions
- ERM is near optimal in this setting.
- NOT DISCUSSED : many interesting problems there are no efficient ERM algorithms. For example linear classification under 0-1 loss. k -literal disjunction learning.

Problems – 1

1. Problem 3.3 in [SS14]
2. Prove $(1 - x) \leq e^{-x}$ for $x \in \mathbb{R}$.
3. Problem 5.3 in [SS14]
4. Problem 6.2 in [SS14]
5. Problem 6.7 in [SS14]
6. On page 49 why did we formulate the VC-dimension of large margin classifiers with respect to a fixed input space X rather than for example $\{\mathbf{x} \in \ell_2 : \|\mathbf{x}\| \leq R\}$?
7. Prove the VC-dimension results on 49.

Suggested Readings

This lecture closely follows Chapters 3,4 and 6 from [SS14] (Chapter 5, for No Free Lunch). We omitted proofs for Sauer's Lemma and the generalisation of our results from finite hypothesis class to classes with bounded VC-dimension. Sauer's Lemma is proved relatively directly in 6.5.1. Extending from finite hypotheses classes to bounded VC-dimension introduces the powerful technique of the double sample trick, proofs via Rademacher Complexities are in 28.1 and 28.3 in [SS14]. Arguably there is more insight to be gained by looking at the proofs that more directly show the parallel to the finite hypothesis class case, I suggest for example Maria-Florina Balcan's notes *Lectures 5,6 and 7 : 8803 Machine Learning Theory 2010*.

Useful references

1. *Understanding Machine Learning from Theory to Algorithms*, S. Shalev-Shwartz and S. Ben-David, Cambridge University Press (2014)

8. Online learning II

COMP0078: Supervised Learning

Mark Herbster

29 November 2021

University College London
Department of Computer Science
Part II Online learning 21v3

Today

- Online Learning with Partial Feedback (Bandits)
- Online Multitask Learning with Long-Term Memory

Bandits

Part I Learning with Partial Feedback

Recall Allocation Setting (HEDGE)

Full Feedback Protocol

For $t = 1$ To m Do

Predict $\hat{y}_t \in [n]$

Observe full loss vector $\ell_t \in [0, 1]^n$

Goal: Design master algorithms with “small regret”.

$$\sum_{t=1}^m \ell_t, \hat{y}_t - \sum_{t=1}^m \min_{i \in [n]} \ell_{t,i} \leq o(m)$$

In the bandit setting we see only feedback for our prediction/action.

Partial Feedback Protocol

For $t = 1$ To m Do

Predict $\hat{y}_t \in [n]$

Observe loss of prediction $\ell_{t, \hat{y}_t} \in [0, 1]$

Goal: as above.

Note: Although (largely) unobserved ℓ_1, \dots, ℓ_m are assumed to exist.

Partial Feedback Setting – Examples

1. **[Online Advertising]:** We have n potential ads to display. If the user click through on the ad we incur 0 loss.
2. **[Medical Trials]:** We have n potential medical treatments. The better the response the less loss.
3. **[Game tree search]:** We have n potential branches to search, if our “evaluation function” increases we proportionally receive less loss.

Exploration and Exploitation

This *Partial Feedback Setting* is often called the *Bandit* setting. The etymology being that we have a slot/fruit machine (once called one-armed bandits) each with potentially different “payback” rates and we wish to play so as to minimise our loss. Metaphorically, we will think of each prediction/action as pulling one of n arms.

Intuitions (Exploration and Exploitation)

1. We need to *use trials* to *explore* by trying different arms to estimate the machine with the minimal expected loss.
2. We need to *use trials* to *exploit* playing the arm with minimal observed loss.
3. *The tradeoff:* the *number of trials* used to *explore* limits *exploitation* and vice versa.

Before considering our “solution” let’s discuss a key tool *Unbiased Estimators*.

Unbiased Estimators

Definition

An **estimator** $\hat{\theta}$ estimates a parameter θ of a distribution from a sample. An estimator is **unbiased** iff $E[\hat{\theta}] = \theta$.

Example 1 (Sample Mean)

Suppose X_1, \dots, X_n are IID random variables from a distribution with mean μ then $\hat{\theta} := \frac{1}{n}(X_1 + \dots + X_n)$ is an unbiased estimator of μ .

Example 2 (German tank problem)

Suppose X is a random variable with the discrete uniform distribution over $\{1, \dots, n\}$.

Suppose n is unknown and we wish to estimate it. The estimator $\hat{\theta}_1 := X$ is the maximum likelihood estimator, since $\mathcal{L}(\theta; X = x) = \frac{1}{\theta}[\theta \geq x]$ which is maximised when $\theta = x$. However for $\hat{\theta}_1$ we have

$$E[\hat{\theta}_1; \theta = n] = \sum_{x=1}^n \frac{1}{n}x = \frac{n+1}{2}$$

However $\hat{\theta}_2 := 2X - 1$ is the unbiased estimator, i.e.,

$$E[\hat{\theta}_2; \theta = n] = \sum_{x=1}^n \frac{1}{n}(2x - 1) = 2 \sum_{x=1}^n \frac{1}{n}x - 1 \sum_{x=1}^n \frac{1}{n} = n + 1 - 1 = n$$

Assumptions and Estimation

Suppose we have a distribution \mathcal{D}_i over $[0, 1]$ for each of $i \in [n]$ arms then each time t we “play” i we receive an IID sample $\ell_{t,i}$ from \mathcal{D}_i . Suppose we play i on trials $S_{t,i} \subseteq [t]$ then $\hat{\mu}_{t,i} := \sum_{t \in S_i} \frac{\ell_{t,i}}{|S_{t,i}|}$ is an unbiased estimator of μ_i .

As we get more samples from arm i the law of large numbers implies $\hat{\mu}_i \rightarrow \mu_i$ and concentration inequalities (e.g., Hoeffding) allow one to quantitatively estimate the likelihood that the estimate differs significantly from the parameter. Using these observations the algorithm UCB balances exploration versus exploitation to obtain good regret bounds for this model. However we would like to consider a more general *adversarial* model. For example suppose \mathcal{D}_i is changing over time (now $\mathcal{D}_{t,i}$) then the estimate $\hat{\mu}_{t,i}$ is biased (now $\hat{\mu}_{t,i} := \frac{\sum_{j=1}^t E[\ell_{j,i}]}{t}$) unless $S_i = [t]$. However if $S_{t,i} = [t]$ then we have no information on the other “arms.”

Needed: a method of obtaining a **simultaneous** unbiased estimate for **all** arms!

Importance Weighting – 1

Suppose X is random variable over \Re with a mean μ . By definition $E[X] = \mu$ and $\hat{\theta}_1 = X$ is an unbiased estimator of the mean. Define the biased coin Z_p with outcome H with probability p and T with probability $(1 - p)$ then define estimator $\hat{\theta}_p$ as equal to X/p if $Z_p = H$ as equal to 0 if $Z_p = T$. Now observe that,

$$E[\hat{\theta}_p] = \mathbb{P}(Z_p = H)(X/p) + \mathbb{P}(Z_p = T)0 = (p)X/p + (1 - p)0 = X$$

and is thus unbiased.

Importance Weighting – 2

We now generalise to obtain an unbiased estimator of ℓ_t in the bandit setting. Given $\mathbf{v}_t \in \Delta_n$ by the notation $\hat{y}_t \sim \mathbf{v}_t$ we mean sample \hat{y}_t from a discrete distribution over $[n]$ where $\mathbb{P}(i) := v_{t,i}$.

Definition (Hallucinated Loss Vector)

We define the unbiased estimator ℓ_t^h of ℓ_t with respect to \mathbf{v}_t as

$$(\ell_{t,i}^h := \frac{\ell_{t,i}}{v_{t,i}}[i = \hat{y}_t])_{i \in [n]},$$

where we have sampled $\hat{y}_t \sim \mathbf{v}_t$.

I.e., ℓ_t^h looks like, $\ell_t^h := (0, 0, \dots, \frac{\ell_{t,\hat{y}_t}}{v_{t,\hat{y}_t}}, 0, \dots, 0)$. Observe that ℓ_t^h is unbiased as for all $i \in [n]$ we have that $E[\ell_{t,i}^h] = \ell_{t,i}$, since

$$E_{\hat{y}_t \sim \mathbf{v}_t} [\ell_{t,i}^h] = \sum_{j=1}^n v_{t,j} \frac{\ell_{t,i}}{v_{t,i}} [i = j] = \ell_{t,i}$$

- Amazingly! we have an unbiased estimator for all arms by only observing a single arm.

The Next Steps

Idea

By sampling a single arm we can obtain an unbiased estimate for ℓ_t . Since we already have an algorithm for the full information setting for arbitrary loss functions (HEDGE) our proposed algorithm is to simply apply HEDGE to the hallucinated loss vectors.

Todo

1. **Problem:** ℓ_t^h is potentially unbounded and HEDGE requires bounded loss vectors. **Fix:** Use a more careful analysis of HEDGE.
2. **Problem:** We will be giving an *expected* regret bound and there will be some subtleties in the sources of randomness. **Fix:** we will clarify the adversarial model that generates ℓ_1, \dots, ℓ_m .

Exp3Parameter $\eta \in (0, \infty)$ Set $\mathbf{v}_1 = (\frac{1}{n}, \dots, \frac{1}{n})$ For $t = 1$ To m do Sample $\hat{y}_t \sim \mathbf{v}_t$ % i.e., treat \mathbf{v}_t as a distribution over $[n]$ Observe loss $\ell_{t, \hat{y}_t} \in [0, 1]$

Construct hallucinated loss vector

$$\ell_t^h := (\ell_{t,i}^h := \frac{\ell_{t,i}}{v_{t,i}}[i = \hat{y}_t])_{i \in [n]}$$

Update

$$v_{t+1,i} = v_{t,i} \exp(-\eta \ell_{t,i}^h) / Z_t \text{ for } i \in [n]$$

$$\text{where } Z_t = \sum_{i=1}^n v_{t,i} \exp(-\eta \ell_{t,i}^h)$$

Footnote: the original EXP3 algorithm was slightly different but it incorporated the key idea of using HEDGE with *hallucinated loss vectors*.

EXP3 – Initial Analysis

Lemma

For any sequence of loss vectors

$$\ell_1, \dots, \ell_m \in [0, 1]^n$$

we have the following inequality for EXP3,

$$\sum_{t=1}^m \mathbf{v}_t \cdot \ell_t^h - \sum_{t=1}^m \mathbf{u} \cdot \ell_t^h \leq \frac{\ln n}{\eta} + \frac{\eta}{2} \sum_{t=1}^m \sum_{i=1}^n v_{t,i} (\ell_{t,i}^h)^2 \text{ for all } \mathbf{u} \in \Delta_n. \quad (1)$$

Proof. The Lemma follows the fact EXP3 is “just” HEDGE with ℓ_t mapped to ℓ_t^h and we proved the above inequality for HEDGE on slide “HEDGE Theorem - Proof (2).”

To finish our analysis:

1. We need to perform expectations so we may replace hallucinated losses ℓ_t^h with the true losses ℓ_t .
2. In order to perform expectations we need to understand the sources of randomness in our model, in particular we need a model for how the “adversary” generates ℓ_1, \dots, ℓ_m
3. Finally we need to bound the term $\sum_{t=1}^m \sum_{i=1}^n v_{t,i} (\ell_{t,i}^h)^2$ and tune η .

Model : Deterministic Oblivious Adversary

Motivation: Our predictions/actions may influence the environment.

Suppose we are using a bandit algorithm to set prices for our products. This may influence the way competitors set prices. If we have an app to display curated news to a user the prior display of new items may effect the user's choices for the next news items. More generally the learners actions may influence the adversary/nature in the future.

Deterministic Oblivious Adversary Model

In this model simply put

$$\ell_1, \dots, \ell_m$$

are determined before the run the algorithm.

However, the process/adversary setting them is assumed to have complete prior knowledge of the learner's algorithm and may set the loss vectors using this knowledge, i.e., if the adversary knows the learner will do (or is likely to) predict something on trial t it may set ℓ_t accordingly. The limitation of this near-omniscient adversary is that it is non-adaptive i.e., if the learner will be taking random actions although the adversary may take this into account in setting ℓ_1, \dots, ℓ_m it cannot however change them "on-the-fly." Observe that this adversary may simulate the *stochastic* model, by simple repeatedly sampling $\mathcal{D}_1, \dots, \mathcal{D}_m$ in advance.

EXP3 - Theorem

Notation: $L_A(S) := \sum_{t=1}^m \ell_{t,\hat{y}_t}$ and $L_i(S) := \sum_{t=1}^m \ell_{t,i}$

Theorem EXP3 (Bound)

[ACFS02]

For any sequence of loss vectors

$$S = \ell_1, \dots, \ell_m \in [0, 1]^n$$

the regret of EXP3 with $\eta = \sqrt{2 \ln n / mn}$ is

$$E[L_A(S)] - \min_i L_i \leq \sqrt{2mn \ln n}.$$

Comparing regrets : HEDGE: $\sqrt{2m \ln n}$ and EXP3: $\sqrt{2mn \ln n}$.

EXP3 Theorem - Proof (1)

Proof – 1

Observe that the only sources of randomness are the samples $\hat{y}_t \sim \mathbf{v}_t$. As previously argued note that $E[\ell_{t,i}^h] = \ell_{t,i}$ and thus

$$E[\mathbf{v}_t \cdot \ell_t^h] = \sum_{i=1}^n E[v_{t,i} \ell_{t,i}^h] = \sum_{i=1}^n v_{t,i} E[\ell_{t,i}^h] = \sum_{i=1}^n v_{t,i} \ell_{t,i} = E[\ell_{t,\hat{y}_t}] \quad (2)$$

and we also have,

$$E[(\ell_{t,i}^h)^2] = \sum_{j=1}^n v_{t,j} \left(\frac{\ell_{t,i}}{v_{t,i}} \right)^2 [j = i]^2 = v_{t,i} \left(\frac{\ell_{t,i}}{v_{t,i}} \right)^2 = \frac{\ell_{t,i}^2}{v_{t,i}} \quad (3)$$

which implies

$$E \left[\sum_{i=1}^n v_{t,i} (\ell_{t,i}^h)^2 \right] = \sum_{i=1}^n v_{t,i} \frac{\ell_{t,i}^2}{v_{t,i}} = \sum_{i=1}^n \ell_{t,i}^2 \leq n. \quad (4)$$

EXP3 Theorem - Proof (2)

Proof – 2

Recalling (1) and taking expectations,

$$E \left[\sum_{t=1}^m \mathbf{v}_t \cdot \ell_t^h - \sum_{t=1}^m \mathbf{u} \cdot \ell_t^h \right] \leq E \left[\frac{\ln n}{\eta} + \frac{\eta}{2} \sum_{t=1}^m \sum_{i=1}^n v_{t,i} (\ell_{t,i}^h)^2 \right] (\forall \mathbf{u} \in \Delta_n)$$

thus,

$$E \left[\sum_{t=1}^m \mathbf{v}_t \cdot \ell_t^h \right] - \min_i E \left[\sum_{t=1}^m \ell_{t,i}^h \right] \leq \frac{\ln n}{\eta} + \frac{\eta}{2} E \left[\sum_{t=1}^m \sum_{i=1}^n v_{t,i} (\ell_{t,i}^h)^2 \right]$$

applying (2) to the first term, lower bounding the second term by observing that \mathbf{u} can be any coordinate vector and that $E[\ell_{t,i}^h] = \ell_{t,i}$ then using (4) for the final term gives

$$E [L_A(S)] - \min_i L_i(S) \leq \frac{\ln n}{\eta} + \frac{\eta}{2} mn$$

now substituting $\eta = \sqrt{2 \ln n / mn}$ proves the theorem. □

Part II

Online Multitask Learning with Long-Term Memory

Eternal Return

*What has been will be again,
what has been done will be done again;
there is nothing new under the sun.*

– Ecclesiastes 1:9

Must not what ever can already have passed this way before? Must not what ever can happen, already have happened, been done, passed by before? And if everything has already been here before, what do you think of this moment, dwarf? Must this gateway too not already – have been here? And are not all things firmly knotted together in such a way that this moment draws after it all things to come? Therefore – itself as well? For, whatever can run, even in this long lane outward-must run it once more! – And this slow spider that creeps in the moonlight, and this moonlight itself, and I and you in the gateway whispering together, whispering of eternal things – must not all of us have been here before? – And return and run in that other lane, outward, before us, in this long, eerie lane – must we not return eternally?

– Friedrich Nietzsche, Thus Spoke Zarathustra

The model



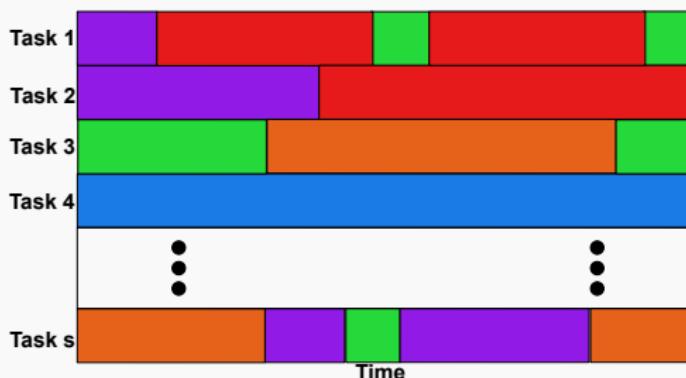
One Mode (“*hypothesis*”) – One Task



Multiple Modes – One Task



Multiple Repeated Modes – One Task



Multiple Modes – Multiple Tasks

Outline

- Single Task Model
 - Finite Hypotheses Classes ("Experts") : \mathcal{H}_n
 - Linear Interpolants with Kernels (RKHS) : \mathcal{H}_K
- Single Task with Switching
- The Share algorithm
 - Switches and Modes
 - Inefficient Algorithms (exponential time)
- Multitask Model
 - Model
 - Algorithm and Bounds (Finite Hypotheses Classes) % Not our focus in this lecture.
 - Algorithm and Bounds (RKHS)
- Online Switching Multitask Learning in an RKHS
 - On-Line (Inductive) Binary Matrix Completion
 - Complexity Measures for Matrices
 - MEG-IMCSI Algorithm and Bound
 - Latent Block Models
 - Side-information
 - Bounding the Quasi-Dimension
 - Reducing Online Switching Multitask Learning to Matrix Completion with Side-information
 - A Simplified Online Multitask Model

One Mode – One Task



One Mode – One Task

On-Line Learning Model

Protocol

For $t = 1$ to T do

Receive pattern $x_t \in \mathcal{X}$

Predict/Act $\hat{y}_t \in \mathcal{Y}$

Receive label $y_t \in \mathcal{Y}$

Incur loss $L_A = L_A + \ell(y_t, \hat{y}_t)$

Evaluation metric: The loss of an Algorithm A on
 $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_T, y_T)$

$$L_A := \sum_{t=1}^T \ell(y_t, \hat{y}_t)$$

Our Goal: Design master algorithms with “small loss”.

How is this possible without distributional assumptions?

Regret Bounds

Idea

We do not give an **absolute** bound but rather a **relative** bound to a **hypothesis** class \mathcal{H} of predictors.

Cumulative loss definitions

$$L_A := \sum_{t=1}^T \ell(y_t, \hat{y}_t) \quad L_h := \sum_{t=1}^T \ell(y_t, h(\mathbf{x}_t))$$

Non-uniform regret bound

$$L_A - L_h \leq \text{regret}(\mathcal{H}, h, T) \quad \forall h \in \mathcal{H}$$

Example Hypotheses Classes

1. $\mathcal{H}_n := \{h^1, \dots, h^n\} \subseteq \{-1, 1\}^{\mathcal{X}}$ “Finite”
2. $\mathcal{H}_K := \overline{\text{span}}(\{K(x, \cdot)\}_{\forall x \in \mathcal{X}})$ \mathcal{H}_K is RKHS def. $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$

0 – 1 Loss Regret Bounds

- Regret Bounds HEDGE/WM/AggA and Online Gradient Descent (“OGD”).
- $\mathcal{L}_{01}(y, \hat{y}) := [y \neq \hat{y}]$, “mistake counting”

Theorem “Hedge/WM/AggA” (Bound) [V90,LW94,FS97]

For any finite hypothesis class $\mathcal{H}_n \subset \{-1, 1\}^{\mathcal{X}}$ and any $(x_1, y_1), \dots, (x_T, y_T)$

$$\sum_{t=1}^T \mathbb{E}[\mathcal{L}_{01}(y_t, \hat{y}_t)] - \mathcal{L}_{01}(y_t, h(x_t)) \leq \sqrt{2 \log(n) T} \quad (\forall h \in \mathcal{H}_n). \quad (5)$$

Theorem “OGD” (Bound) [CLW96,S11]

For any kernel K and any $(x_1, y_1), \dots, (x_T, y_T)$

$$\sum_{t=1}^T \mathbb{E}[\mathcal{L}_{01}(y_t, \hat{y}_t)] - \mathcal{L}_{01}(y_t, h(x_t)) \leq \frac{1}{2} \sqrt{\|h\|_K^2 X^2 T} \quad (\forall h \in \mathring{\mathcal{H}}_K) \quad (6)$$

with $X^2 := \max_{t \in [T]} K(x_t, x_t)$ and interpolating hypotheses

$$\mathring{\mathcal{H}}_K := \{h \in \mathcal{H}_K : h(x_t) \in \{-1, 1\}, \forall t \in [T]\}.$$

- Bounds depend on tuning of parameters for algorithms.
- **Note:** difference between linear *classification* and *interpolation*.
- No known eff. algs. achieve regret bounds for linear classification.

Hedge/WM/AggA

Parameters: Learning rate $\gamma \in [0, \infty)$; finite hypothesis set

$$\{h^1, \dots, h^n\} = \mathcal{H}_n \subset \{-1, 1\}^{\mathcal{X}}$$

Initialization: Initialize $v_1 = \frac{1}{n}\mathbf{1}$

For $t = 1, \dots, T$

- Receive instance $\mathbf{x}_t \in \mathcal{X}$.
- Set $\mathbf{h}_t = (h^1(\mathbf{x}_t) \dots h^n(\mathbf{x}_t)) \in \{-1, 1\}^n$.
- Predict

$$i_t \sim v_t; \hat{y}_t \leftarrow h_t^{i_t}.$$

- Receive label $y_t \in \{-1, 1\}$.
- Update

$$\ell_t \leftarrow \frac{1}{2} |\mathbf{h}_t - \hat{y}_t \mathbf{1}|$$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t \odot \exp(-\gamma \ell_t)$$

$$\mathbf{v}_{t+1} \leftarrow \frac{\mathbf{w}_{t+1}}{\sum_{i=1}^n w_{t+1,i}}$$

Algorithm: HEDGE/WM

Randomised Kernel OGD

Parameters: Learning rate $\gamma \in [0, \infty)$, SPD Kernel $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$

Initialization: Initialize $\mathbf{w}_1 = \mathbf{0}$

For $t = 1, \dots, T$

- Receive $x_t \in \mathcal{X}$.
- Predict

$$Y_t \sim \text{UNIFORM}(-1, 1); \bar{y}_t \leftarrow \mathbf{w}_t(x_t); \hat{y}_t \leftarrow \text{sign}(\bar{y}_t - Y_t).$$

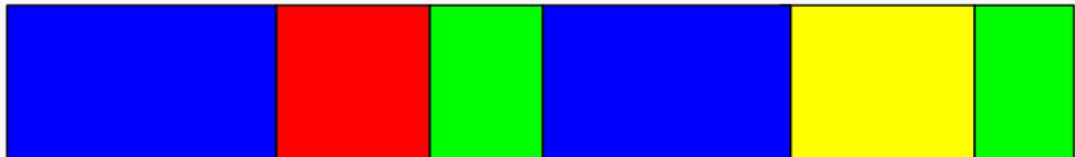
- Receive label $y_t \in \{-1, 1\}$.
- If $\bar{y}_t y_t \leq 1$ then

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \gamma y_t K(x_t, \cdot)$$

- Else $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$

Algorithm: Randomized Online Kernel Gradient Descent (OGD)

Switching – Single Task



Switching – Single Task

Share Algorithm

Learning with sequences of experts

On-line Learning (Review)

time t	1	2	3	4	...	t
expert 1	.5	3	2	1	...	$x_{t,1}$
expert 2	.75	-1.5	-1	-2	...	$x_{t,2}$
expert 3	-1.5	3	2	-4	...	$x_{t,3}$
expert 4	.75	-1.3	1.5	1.5	...	$x_{t,4}$
alg. preds	0	1.5	1.5	.75	...	\hat{y}_t
label	.75	-1.5	2	1	...	y_t
alg. loss	0.56	9	.25	.06	...	$(\hat{y}_t - y_t)^2$

For a sequence of examples $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_T, y_T)\}$

$$\text{Loss}_A(S) = \sum_{t=1}^T (\hat{y}_t - y_t)^2$$

Aim: to bound $\text{Loss}_A(S)$ in terms of the loss of the best expert.

$$\text{Loss}_i(S) = \sum_{t=1}^T (x_{t,i} - y_t)^2$$

Static Relative Loss Bounds

For all sequences of examples S

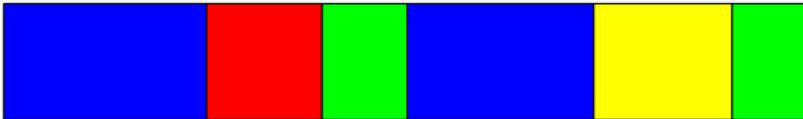
$$\text{Loss}_A(S) \leq \text{Loss}_i(S) + c \ln n \quad [V90, LW94, HKW98]$$

- The loss of the algorithm is bounded in terms of the best **single** expert.

So far the comparator is “static”

Recall: 1) That c depends on the loss function for example log ($c = 1$) and square ($c = 2$). In the allocation model the results will still apply but we will gain an additional term that depend $\Theta(\sqrt{\log nT})$.

Switching Model



Sequence of hypotheses (switches $k(h^*) = 5$, modes $m(h^*) = 4$)

Let $h^* := (h_1, h_2, \dots, h_T)$ denote a sequence of hypotheses.

Loss for a sequence of hypotheses

$$\mathsf{L}_{h^*} = \sum_{t=1}^T \ell(y_t, h_t(x_t))$$

Measures of complexity for a sequence of hypotheses

$$k(h^*) := \underbrace{\sum_{t=1}^T [h_t \neq h_{t+1}]}_{\text{switches}} ; \quad m(h^*) := \underbrace{\bigcup_{t=1}^T \{h_t\}}_{\text{modes (distinct hypotheses)}}$$

Note: Switch times and constituent hypotheses are **unknown** to the learner.

Multiple Modes

Abbreviation: Set $k := \text{size}(\{h_1, h_2, \dots, h_T\})$.

Proposition

The FIXED-SHARE algorithm achieves,

1. $\mathcal{X} = [0, 1]^n$
2. $\mathcal{Y} = [0, 1]$
3. $\ell(y_t, \hat{y}_t) = y \log \frac{y}{\hat{y}} + (1 - y) \log \frac{1-y}{1-\hat{y}}$
4. $\mathcal{H}_e := \{h_i : i = 1, \dots, n\}$ with $h_i(\mathbf{x}) := x_i$
- 5.

$$L_A - L_{\{h_1, h_2, \dots, h_T\}} \leq k \log \frac{T-1}{k} + k \log(n-1) + \log n + k$$

$$\forall \{h_1, h_2, \dots, h_T\} \subset \mathcal{H}_e$$

Asymptotically we pay $\log \frac{T}{k} + \ln(n-1)$ per mode.

Fixed Share Algorithm

- **Parameters:** $0 \leq \eta$ and $0 \leq \alpha \leq 1$.
- **Initialization:** Set the weights to $w_{1,1}^s = \dots = w_{1,n}^s = \frac{1}{n}$.
- **Prediction:** Let $v_{t,i} = \frac{w_{t,i}^s}{W_t}$, where $W_t = \sum_{i=1}^n w_{t,i}^s$. Predict with $\hat{y}_t = \mathbf{v}_t \cdot \mathbf{x}_t$
- **Loss Update:** After receiving the t th outcome y_t ,

$$\forall i : 1, \dots, n : w_{t,i}^m = w_{t,i}^s e^{-\eta \ell(y_t, x_{t,i})}$$

- **Fixed Share Update:**

- $\text{pool} = \sum_{i=1}^n \alpha w_{t,i}^m$
- $\forall i : 1, \dots, n : w_{t+1,i}^s = (1 - \alpha) w_{t,i}^m + \frac{1}{n-1} w_{t,i}^s (\text{pool} - \alpha w_{t,i}^m)$

Shifting Loss Bounds

Shifting Experts:

- Let

$$k := \text{size}(\{i_1, i_2, \dots, i_T\}) := \sum_{k=1}^{T-1} [i_k \neq i_{k+1}]$$

- $L^* := \text{Loss}_{\{i_1, i_2, \dots, i_T\}}(S)$
- Choose $\alpha \in [0, 1]$,

We then have the bound:

$$L_A(S) \leq L^* + c[(T-1)(H(\alpha^*) + D(\alpha^* \parallel \alpha)) + k \ln(n-1) + \ln n],$$

where $\alpha^* = \frac{k}{T-1}$, $H(p) = \frac{1}{p} \ln \frac{1}{p} + \frac{1}{1-p} \ln \frac{1}{1-p}$ and

$$D(p^* \parallel p) = p^* \ln \frac{p^*}{p} + (1 - p^*) \ln \frac{1 - p^*}{1 - p}$$

When $\alpha = \frac{k}{T-1}$, then the above is upper bounded by

$$L_A(S) \leq L^* + c[k \ln \frac{T-1}{k} + k \ln(n-1) + \ln n + k],$$

Shifting Loss Bounds (Proof Sketch – 1)

Proof Sketch – 1

1. Observe that the bound $\text{Loss}_A(S) \leq \text{Loss}_i(S) + c \ln n$ for the expert setting trivially generalises to

$$\text{Loss}_A(S) \leq \text{Loss}_i(S) + c \ln \frac{1}{w_{1,i}} \quad (i \in [n])$$

i.e., rather than assign all experts uniform probability initially we can choose an arbitrary prior weights (probability).

2. Consider the following set of n^T initial weights:

$$w_{1,\{i_1, i_2, \dots, i_T\}} := \frac{1}{n} (1 - \alpha)^{T-1-k} \left(\frac{\alpha}{n-1} \right)^k \quad (\{i_1, i_2, \dots, i_T\} \in [n]^T)$$

3. Observe that the sum of these weights is 1.
4. These are just “meta-experts” that describe all possible expert sequences over T trials.

Shifting Loss Bounds (Proof Sketch – 2)

Proof Sketch – 2

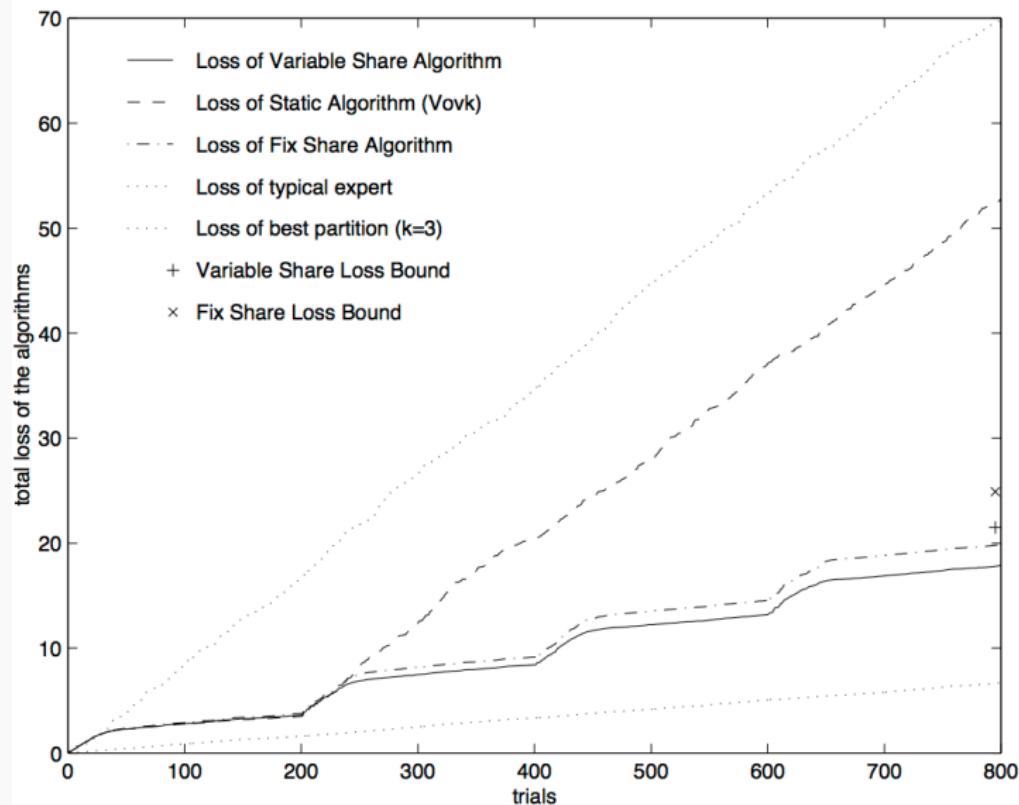
1. Observe if we apply the WA algorithm with these meta-experts then the previous bound follows (with some algebra) by just simplifying $\ln \frac{1}{w_{1,\{i_1, i_2, \dots, i_T\}}}$.
2. With a more detailed analysis one can then show FIXED-SHARE algorithm perfectly simulates the prediction of the algorithm with T^n meta-experts in $O(n)$ time.

□

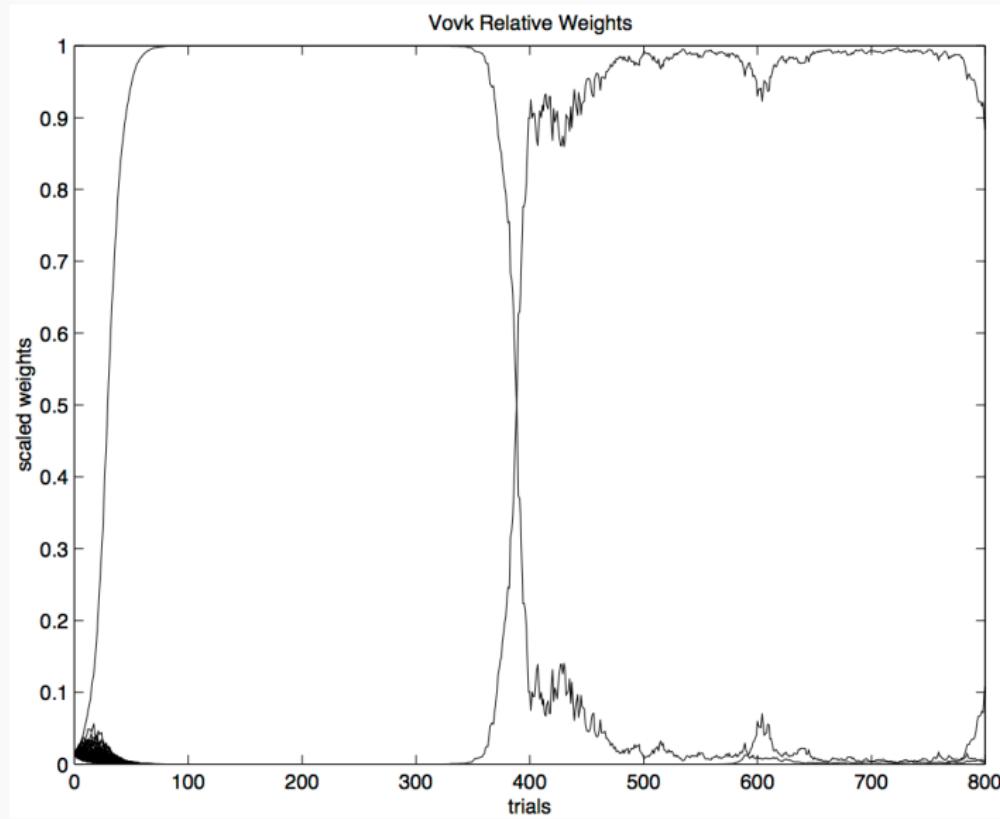
Experiment (simulation)

1. Trials (T) : 800
2. Shifts(k): 3
3. Number of Experts (n) : 64
4. Loss function (L): $L(y, \hat{y}) = (y - \hat{y})^2$ ($c = 1/2, \eta = 2$)
5. Share Parameter(α) : 0.024
6. Typical expert expected Loss/Trial : 1/12
7. “Best” expert expected Loss/Trial : 1/120

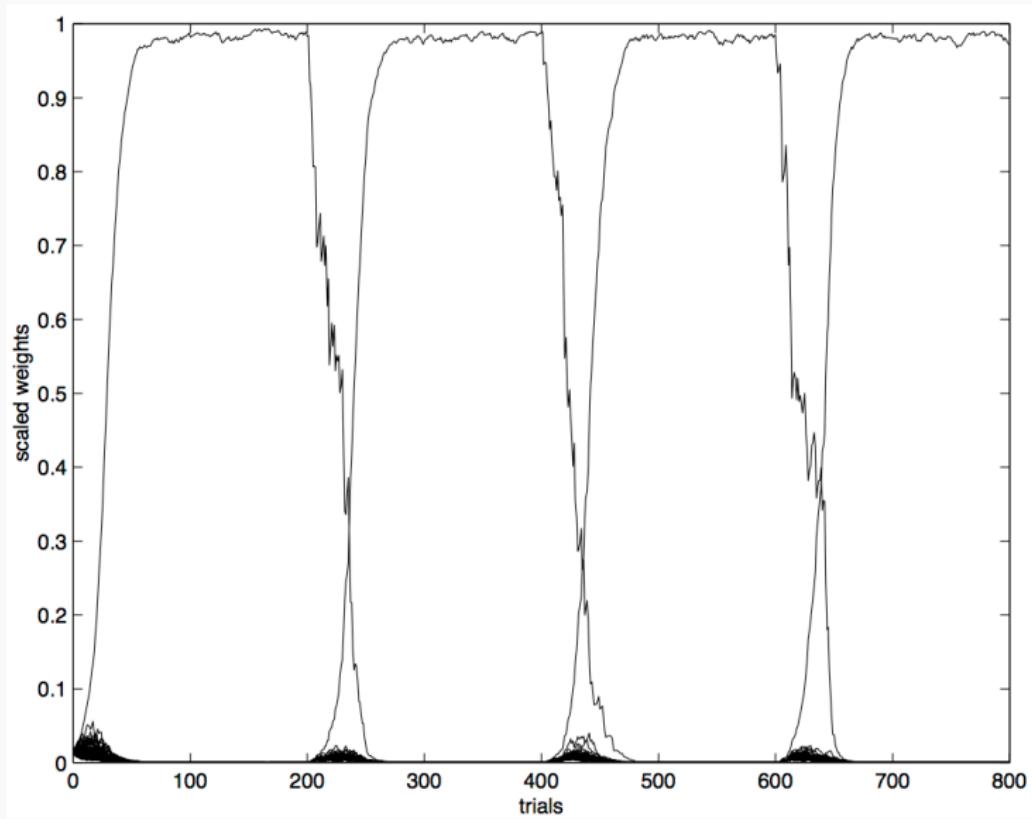
Share Algorithms – Performance



Expert Algorithm's Weights



Variable Share Weights



Some Applications

1. Predicting disk idle times
2. Online load balancing (process migration determination)
3. Predicting TCP packet inter-arrival times
4. Financial prediction by combining portfolios
5. Combining language for domain and topic adaptation

Introducing Memory

Learning a sequences of hypotheses paying less when a hypotheses “returns.”

Reasoning about upper bounds for switching for \mathcal{H}_n (MEMORY)

1. Reduce switching regret via creating “meta”-hypotheses \bar{h} from \mathcal{H}_n

$$\mathcal{H}_n^{T,k,m} := \{\bar{h} \in \mathcal{H}^T : k(\bar{h}) = k, |m(\bar{h})| = m\}$$

2. A meta-hypothesis “colors” each trial t with a hypothesis from \mathcal{H}_n .

3. Bounding $|\mathcal{H}_n^{T,k,m}| \leq \binom{n}{m} \binom{T-1}{k} m^{k+1}$;

$$\log |\mathcal{H}_n^{T,k,m}| \leq m \log \frac{n}{m} + k \log \frac{T-1}{k} + (k+1) \log m + k + m$$

4. Recalling the regret bound of HEDGE,

$$\mathbb{E}[L_{\text{HEDGE}}] - L_h \leq \sqrt{2T \log(n)} \quad (\forall h \in \mathcal{H}_n).$$

5. Bounds immediately follow but algorithms are exponential in k & m .

6. An $\mathcal{O}(n)$ algorithm for $\mathcal{H}_n^{T,k,m}$ was posed as an open problem in [F00] solved by [BW03] and improved in [KAW12].



Meta-hypotheses \bar{h} with $k = 5$ and $m = 4$.

Reasoning about upper bounds for switching for $\mathcal{H}_K - 1$

1. Previous approach fails since $|\mathcal{H}_K|$ is typically infinite.
2. Meta-hypotheses are now meta-“learners” \hat{h} .
3. A meta-learner \hat{h} consists of m instances of OGD.
4. The “color” of a trial is the associated OGD instance.
5. An OGD instance only runs only its colored trials with state saved/restored between trials.

Start OGD-1	Cont. OGD-1	Cont. OGD-1	Start OGD-2	Cont. OGD-1	Start OGD-3	Cont. OGD-3	Cont. OGD-3	Start OGD-4	Cont. OGD-4	Cont. OGD-1
t=1	2	3	4	5	6	7	8	9	10	11

A meta-learner \hat{h} with $k = 5$ and $m = 4$.

Reasoning about upper bounds for switching for \mathcal{H}_K - 2

1. Thus the loss of a single meta-learner \hat{h} is

$$L_{\hat{h}} = \sum_{i=1}^m L_{\text{OGD}-i}$$

OGD- i receives a subsequence from a partition of trial sequence (**POTS**),

$$(x_1(i), y_1(i)), \dots, (x_{T_i}(i), y_{T_i}(i)) \text{ with } T = \sum_{i=1}^m T_i.$$

2. Using OGD bounding via (6)

$$R = \sum_{i=1}^m \sum_{t=1}^{T_i} \mathbb{E}[\mathcal{L}_{01}(y_t(i), \hat{h}_t(i))] - \mathcal{L}_{01}(y_t, h_i(x_t(i))) \leq \frac{1}{2} \sum_{i=1}^m \sqrt{\|h_i\|_K^2 X^2 T_i}$$

for a fixed **POTS** with any $h_1, \dots, h_m \in \mathcal{H}_K^\circ$

3. Then using HEDGE to mix over all \hat{h} bounding via (5)

$$R \leq \mathcal{O} \left(\sqrt{\left(\sum_{i=1}^m \|h_i\|_K^2 X^2 + k \log m + k \log \frac{T}{k} \right) T} \right)$$

for any **POTS** with any $h_1, \dots, h_m \in \mathcal{H}_K^\circ$.

Considerations

For finite hypotheses classes we obtained,

$$R \leq \mathcal{O} \left(\sqrt{\left(m \log \frac{n}{m} + k \log m + k \log \frac{T}{k} \right) T} \right) \quad (\forall \bar{h} \in \mathcal{H}_n^{T,k,m}) \quad (7)$$

and for RKHS hypotheses classes we obtained,

$$R \leq \mathcal{O} \left(\sqrt{\left(\sum_{i=1}^m \|h_i\|_K^2 X^2 + k \log m + k \log \frac{T}{k} \right) T} \right) \quad (\forall \bar{h} \in \mathring{\mathcal{H}}_K^{T,k,m}) \quad (8)$$

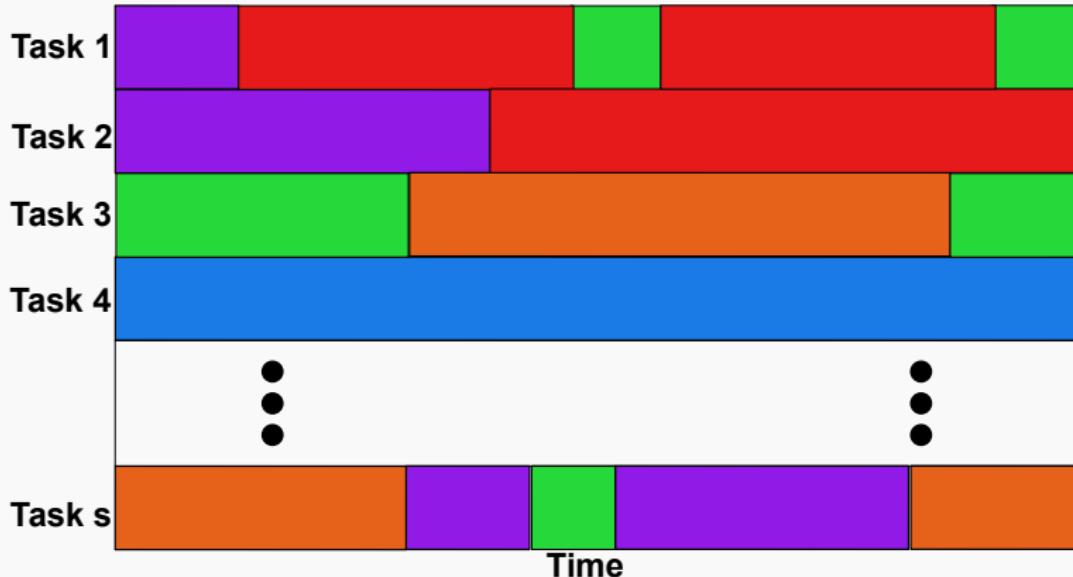
- We view the blue terms as “learner complexities” and the remaining terms as the price for their sequence locations.

GOAL: to obtain these bounds with *efficient* algorithms.

A Novel Model

Multiple Tasks

Multiple Tasks with Switching and Memory



Multiple Tasks with Switching and Memory

Multiple Task Learning Model

Protocol

For $\tau = 1$ to T do

 Receive task $i \in [s]$; set t to "local time" in task i

 Receive pattern $x_t^i \in \mathcal{X}$

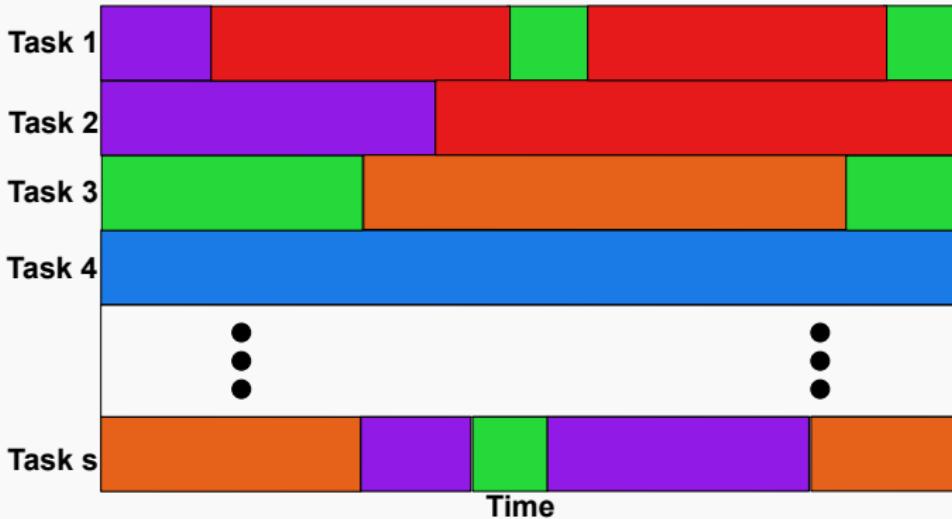
 Predict/Act $\hat{y}_t^i \in \mathcal{Y}$

 Receive label $y_t^i \in \mathcal{Y}$

 Incur loss $L_A = L_A + \ell(y_t^i, \hat{y}_t^i)$

Notation: Global time is τ and local time is t wrt task i .

Measures of Complexity



$$k(h^*) := \text{switches} = \sum_{i=1}^s \sum_{t=1}^{T-1} [h_t^i \neq h_{t+1}^i] \quad m(h^*) := \text{modes} = \bigcup_{i,t}^s \{h_t^i\}$$

- Definitions are direct extension of single task case.
- Combinatorically tasks *reduce* to s additional switches.
- Algorithmically tasks *do not reduce* to switches.

Theorem (Finite Hypotheses Classes)

Theorem [HPT20b] (Finite Hypotheses Classes)

There exists an algorithm with parameters $\mathcal{H}_n \subseteq \{-1, 1\}^{\mathcal{X}}$; $s, m, k, T \in \mathbb{N}$ and

$$C \leq m \log \left(\frac{n}{m} \right) + s(\log(m) + 1) + k \left(\log(m - 1) + 2 \log \left(\frac{T - s}{k} \right) + 2 \right).$$

with expected regret bounded above by

$$\sum_{i=1}^s \sum_{t=1}^{T^i} \mathbb{E}[\mathcal{L}_{01}(y_t^i, \hat{y}_t^i)] - \mathcal{L}_{01}(y_t^i, h_t^i(x_t^i)) \leq \sqrt{2CT} \quad (9)$$

for any $\mathbf{h}^* \in \mathcal{H}_n^T$ such that $k = k(\mathbf{h}^*)$, $m \geq |m(\mathbf{h}^*)|$, $m > 1$.

- The algorithm and analyses is directly based on the “Markov circadian specialist” method of [KAW12].
- We match (7) up to constant factors in the single (multi-) task case.

Predicting \mathcal{H}_n in a switching multitask setting.

Parameters: $\mathcal{H}_n \subseteq \{-1, 1\}^{\mathcal{X}}$; $s, m, k, T \in \mathbb{N}$

Initialization: $n := |\mathcal{H}_n|$; $\pi^1 \leftarrow \frac{1}{n}$; $\mu := \frac{1}{m}$; $\mathbf{w}_1^1 = \dots = \mathbf{w}_1^s \leftarrow \mu \mathbf{1}$; $\theta := 1 - \frac{k}{T-s}$; $\phi := \frac{k}{(m-1)(T-s)}$ and
 $\eta := \sqrt{\left(m \log\left(\frac{n}{m}\right) + smH\left(\frac{1}{m}\right) + (T-s)H\left(\frac{k}{T-s}\right) + (m-1)(T-s)H\left(\frac{k}{(m-1)(T-s)}\right)\right) \frac{2}{T}}$

For $\tau = 1, \dots, T$

- Receive task $\ell^\tau \in [s]$.
- Receive $x^\tau \in \mathcal{X}$.
- Set $i \leftarrow \ell^\tau$; $t \leftarrow \sigma(\tau)$. % Convert “global time” τ to “local time” $\frac{i}{t}$.
- Predict

$$\mathbf{v}^\tau \leftarrow \frac{\pi^\tau \odot \mathbf{w}_t^i}{\pi^\tau \cdot \mathbf{w}_t^i}, \quad \hat{h}^\tau \sim \mathbf{v}^\tau, \quad \hat{y}^\tau \leftarrow \hat{h}^\tau(x^\tau).$$

- Receive $y^\tau \in \{-1, 1\}$.
- Update:

i) $\forall h \in \mathcal{H}_n, c_h^\tau = \mathcal{L}_{01}(h(x^\tau), y^\tau)$	ii) $\delta \leftarrow \mathbf{w}_t^i \odot \exp(-\gamma \mathbf{c}^\tau)$
iii) $\beta \leftarrow (\pi^\tau \cdot \mathbf{w}_t^i) / (\pi^\tau \cdot \delta)$	iv) $\epsilon \leftarrow \mathbf{1} - \mathbf{w}_t^i + \beta \delta$
v) $\pi^{\tau+1} \leftarrow \pi^\tau \odot \epsilon$	vi) $\mathbf{w}_{t+1}^i \leftarrow (\phi(\mathbf{1} - \mathbf{w}_t^i) + \theta \beta \delta) \odot \epsilon^{-1}$

Algorithm: Predicting \mathcal{H}_n in a switching multitask setting.

- Requires $\mathcal{O}(n)$ time per trial to predict.
- Maintains a global weight vector $\pi \in \Delta_n$ and task vectors $\mathbf{w}^1, \dots, \mathbf{w}^s \in [0, 1]^n$.

Theorem (RKHS Hypotheses Classes)

Theorem [HPT20b] (RKHS Hypotheses Classes)

There exists an algorithm with upper parameter estimates,
 $k \geq k(\mathbf{h}^*)$, $m \geq |m(\mathbf{h}^*)|$,

$$\hat{C} \geq C(\mathbf{h}^*) := \left(\sum_{h \in m(\mathbf{h}^*)} \|h\|_K^2 X_K^2 + 2(s+k-1)m[\log_2 T]^2 + 2m^2 \right),$$

$\hat{X}_K^2 \geq \max_{\tau \in [T]} K(x^\tau, x^\tau)$, and learning rate $\gamma = \sqrt{\frac{\hat{C} \log(2T)}{2Tm}}$ is bounded by

$$\sum_{i=1}^s \sum_{t=1}^{T^i} \mathbb{E}[\mathcal{L}_{01}(y_t^i, \hat{y}_t^i)] - \mathcal{L}_{01}(y_t^i, h_t^i(x_t^i)) \leq 4\sqrt{2\hat{C} T \log(2T)} \quad (10)$$

for any $\mathbf{h}^* \in \mathring{\mathcal{H}}_K^T$.

- Next section we provide a few intuitions about the algorithm.

Considerations

In the single task case ($s = 1$) for \mathcal{H}_n in (9) we obtained,

$$R \leq \mathcal{O} \left(\sqrt{\left(m \log \frac{n}{m} + k \log m + k \log \frac{T}{k} \right) T} \right)$$

which matches (7) up to constant factors.

For \mathcal{H}_K we obtain,

$$R \leq \mathcal{O} \left(\sqrt{\left(\sum_{i=1}^m \|h_i\|_K^2 X^2 + k \log m + k \log \frac{T}{k} \right) T} \right) \quad [\text{Exponential time in } m]$$

Whereas with an efficient alg. (Inductive Matrix Completion) we obtain

$$R \leq \mathcal{O} \left(\sqrt{\left(\sum_{i=1}^m \|h_i\|_K^2 X^2 + km(\log T)^2 \right) T \log T} \right) \quad [\mathcal{O}(T^3) \text{ time per trial}]$$

- Thus our efficient algorithm gains poly-log factors in T and we now scale with m rather than $\log m$.
- In many cases the dominant term is the “learner complexity” in red.

Methods

Online Switching Multitask Learning in an RKHS – Overview

- **Idea:** Reduce Online Multitask Learning to Matrix Completion.
 1. Inductive Matrix Completion with Side-Information
 2. MEG-IMCSI Algorithm

Matrix Completion

		Movies							
		Star Wars	Jules & Jim	Mamma Mia					
Users		Jane	5	4		5	5	5	4
		Joe			1	3	3	2	
				1		3			
			1	2			5	2	
					3		1		3
				1	5	3	2		
		Xerxes	4	2			1	2	2

- **Matrix completion:** fill in the missing values.
- For example, the rows may represent **users** and the columns **movies**.
- Netflix had a \$1,000,000 challenge with a matrix consisted of $480,189 \text{ users} \times 17,770 \text{ users}$ with 100,480,507 rating – won in 2009.

On-Line (Inductive) Binary Matrix Completion

Protocol

For $t = 1$ to T do

Receive indices $(i_t, j_t) \in \mathcal{I} \times \mathcal{J}$

Predict/Act $\hat{y}_t \in \{-1, 1\}$

Receive label $y_t \in \{-1, 1\}$

Incur loss $L_A = L_A + \mathcal{L}_{01}(y_t, \hat{y}_t)$

- **Side Information:** Kernels $\mathcal{M}^+ : \mathcal{I} \times \mathcal{I} \rightarrow \mathbb{R}$, $\mathcal{N}^+ : \mathcal{J} \times \mathcal{J} \rightarrow \mathbb{R}$.

- **Aim:** Regret bounds of the form,

$$\sum_{t=1}^T \mathbb{E}[\mathcal{L}_{01}(y_t, \hat{y}_t)] - \mathcal{L}_{01}(y_t, U_{i_t, j_t}) \leq \text{regret}(\mathbf{U}, \mathcal{M}^+, \mathcal{N}^+, T) \quad (\forall \mathbf{U} \in \{-1, 1\}^{\mathcal{I} \times \mathcal{J}})$$

- **Note:** The index sets \mathcal{I} and \mathcal{J} may potentially be infinite.

- **Example:** Consider the case of users and movies where associated with each is a feature vector. In the inductive model the side information allows for non-trivial predictions for a user or movie without any observations for that particular movie or user.

Matrix Complexity

Rank Complexity

A natural notion of complexity for matrices is that of a low rank decomposition. A matrix $\mathbf{U} \in \mathbb{R}^{m \times n}$ has a rank- k decomposition if there exists $\mathbf{P} \in \mathbb{R}^{m \times k}$ and $\mathbf{Q} \in \mathbb{R}^{n \times k}$ such that $\mathbf{U} = \mathbf{P}\mathbf{Q}^\top$. Observe that a rank- k decomposition is specified by $k(m + n)$ parameters.

Factor Models

Consider the Netflix problem. A very simple model is that for each user i we associate a factor p_i which is how positive they are about movies and likewise for movies for each movie j we associate a factor q_j which is popular the movie and thus for any particular (user i , movie j) the score associated is just $U_{ij} = p_i \times q_j$ this is a rank-1 decomposition $\mathbf{p} \in \mathbb{R}^n$ and $\mathbf{q} \in \mathbb{R}^m$. This is an overly simple model ... slightly more complex we might consider there are k factors and each user and movie has a score with respect to these factor (e.g., comedy, romance, age, education) and total score is the sum of the scores i.e., $U_{ij} = \sum_{s=1}^k p_i^{(s)} q_j^{(s)}$, i.e., a rank- k decomposition.

Complexity Measures for Matrices

Max-norm of $\mathbf{U} \in \mathbb{R}^{m \times n}$

$$\|\mathbf{U}\|_{\max} := \min_{\mathbf{PQ}^T = \mathbf{U}} \left\{ \max_{1 \leq i \leq m} \|\mathbf{P}_i\| \times \max_{1 \leq j \leq n} \|\mathbf{Q}_j\| \right\},$$

where the minimum is over all matrices $\mathbf{P} \in \mathbb{R}^{m \times d}$, $\mathbf{Q} \in \mathbb{R}^{n \times d}$ and every integer d . **Note:** $1 \leq \|\mathbf{U}\|_{\max} \leq \min(\sqrt{m}, \sqrt{n})$

Quasi-dimension of $\mathbf{U} \in \mathbb{R}^{m \times n}$ wrt $\mathbf{M} \in \mathbf{S}_{++}^m$, $\mathbf{N} \in \mathbf{S}_{++}^n$ at margin γ

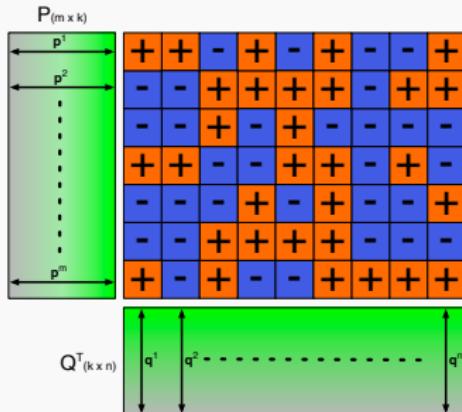
$$\mathcal{D}_{\mathbf{M}, \mathbf{N}}^\gamma(\mathbf{U}) := \min_{\hat{\mathbf{P}} \hat{\mathbf{Q}}^T = \gamma \mathbf{U}} \mathcal{R}_{\mathbf{M}} \operatorname{tr}(\hat{\mathbf{P}}^T \mathbf{M} \hat{\mathbf{P}}) + \mathcal{R}_{\mathbf{N}} \operatorname{tr}(\hat{\mathbf{Q}}^T \mathbf{N} \hat{\mathbf{Q}}),$$

where the infimum is over all row-normalized matrices $\hat{\mathbf{P}} \in \mathcal{N}^{m,d}$ and $\hat{\mathbf{Q}} \in \mathcal{N}^{n,d}$ and every integer d . If the infimum does not exist then $\mathcal{D}_{\mathbf{M}, \mathbf{N}}^\gamma(\mathbf{U}) := +\infty$. If \mathbf{M}, \mathbf{N} are identity matrices then

$$\mathcal{D}_{\mathbf{M}, \mathbf{N}}^\gamma(\mathbf{U}) = m + n$$

Where \mathbf{S}_{++}^m is the set of strictly positive definite matrices $m \times m$ matrices, $\mathcal{R}_{\mathbf{M}} := \max_{i \in [m]} M_{ii}^+$, $\mathcal{N}^{m,d} := \{\hat{\mathbf{P}} \subset \mathbb{R}^{m \times d} : \|\hat{\mathbf{P}}_i\| = 1, i \in [m]\}$ is the set of $m \times d$ row-normalized matrices, and \mathbf{P}_i is the i th row of \mathbf{P} .

Max Norm – Illustrated



Definition:

$$\|U\|_{\max} := \min_{\substack{P \in \mathbb{R}^{m \times k} \\ Q \in \mathbb{R}^{n \times k} \\ k \in \mathbb{N}}} \max_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}} \|p_i\|_2 \|q_j\|_2$$

$$\forall r, s: (PQ^T)_{rs} = U_{rs}$$

Expected Regret MEG-IMCSI Algorithm

Theorem [HPT20a]

The expected regret of the MEG-IMCSI algorithm with parameters

$\gamma \in (0, 1]$, $\widehat{\mathcal{D}} \geq \mathcal{D}_{\mathbf{M}, \mathbf{N}}^{\gamma}(\mathbf{U})$, $\gamma = \sqrt{\frac{\widehat{\mathcal{D}} \log(m+n)}{2T}}$, p.d. matrices $\mathbf{M} \in \mathbf{S}_{++}^m$ and $\mathbf{N} \in \mathbf{S}_{++}^n$ is bounded by

$$\sum_{t \in [T]} \mathbb{E}[\mathcal{L}_{01}(y_t, \hat{y}_t)] - \mathcal{L}_{01}(y_t, \mathbf{U}_{i_t j_t}) \leq 4 \sqrt{2 \frac{\widehat{\mathcal{D}}}{\gamma^2} \log(m+n) T}$$

for all $\mathbf{U} \in \{-1, 1\}^{m \times n}$ with $\|\mathbf{U}\|_{max} \leq 1/\gamma$.

Note: Proof builds on techniques from [TRW05, SSSHZ15, GHP13].

Bounds for online matrix completion with general loss functions and with out side-information were first given in [HKS12].

MEG-IMCSI Algorithm [HPT20a]

Parameters: Learning rate: $0 < \gamma$ quasi-dimension estimate: $1 \leq \lambda$, margin estimate: $0 < \gamma \leq 1$ and side-information kernels $\mathcal{M}^+ : \mathcal{I} \times \mathcal{I} \rightarrow \mathbb{R}$, $\mathcal{N}^+ : \mathcal{J} \times \mathcal{J} \rightarrow \mathbb{R}$, with $\mathcal{R}_{\mathcal{M}} := \max_{i \in \mathcal{I}} \mathcal{M}^+(i, i)$, $\mathcal{R}_{\mathcal{N}} := \max_{j \in \mathcal{J}} \mathcal{N}^+(j, j)$.

Initialization: $\mathbb{M} \leftarrow \emptyset$, $\mathbb{U} \leftarrow \emptyset$, $\mathcal{I}^1 \leftarrow \emptyset$, $\mathcal{J}^1 \leftarrow \emptyset$.

For $t = 1, \dots, T$

- Receive pair $(i_t, j_t) \in \mathcal{I} \times \mathcal{J}$.
- Define

$$(\mathbf{M}^t)^+ := (\mathcal{M}^+(i_r, i_s))_{r, s \in \mathcal{I}^t \cup \{i_t\}} ; \quad (\mathbf{N}^t)^+ := (\mathcal{N}^+(j_r, j_s))_{r, s \in \mathcal{J}^t \cup \{j_t\}},$$

$$\tilde{\mathbf{X}}^t(s) := \left[\frac{(\sqrt{(\mathbf{M}^t)^+}) \mathbf{e}^{is}}{\sqrt{2\mathcal{R}_{\mathcal{M}}}} ; \frac{(\sqrt{(\mathbf{N}^t)^+}) \mathbf{e}^{js}}{\sqrt{2\mathcal{R}_{\mathcal{N}}}} \right] \left[\frac{(\sqrt{(\mathbf{M}^t)^+}) \mathbf{e}^{is}}{\sqrt{2\mathcal{R}_{\mathcal{M}}}} ; \frac{(\sqrt{(\mathbf{N}^t)^+}) \mathbf{e}^{js}}{\sqrt{2\mathcal{R}_{\mathcal{N}}}} \right]^{\top},$$

$$\tilde{\mathbf{W}}^t \leftarrow \exp \left(\log \left(\frac{\lambda}{m+n} \right) I^{|\mathcal{I}^t| + |\mathcal{J}^t| + 2} + \sum_{s \in \mathbb{U}} \gamma y_s \tilde{\mathbf{X}}^t(s) \right).$$

- Predict

$$Y_t \sim \text{UNIFORM}(-\gamma, \gamma); \bar{y}_t \leftarrow \text{tr}(\tilde{\mathbf{W}}^t \tilde{\mathbf{X}}^t) - 1; \hat{y}_t \leftarrow \text{sign}(\bar{y}_t - Y_t).$$

- Receive label $y_t \in \{-1, 1\}$.

- If $y_t \bar{y}_t \leq \gamma$ then

$$\mathbb{U} \leftarrow \mathbb{U} \cup \{t\}, \quad \mathcal{I}^{t+1} \leftarrow \mathcal{I}^t \cup \{i_t\}, \text{ and } \mathcal{J}^{t+1} \leftarrow \mathcal{J}^t \cup \{j_t\}.$$

- Else $\mathcal{I}^{t+1} \leftarrow \mathcal{I}^t$ and $\mathcal{J}^{t+1} \leftarrow \mathcal{J}^t$.

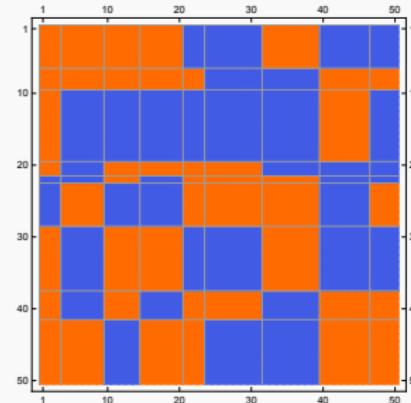
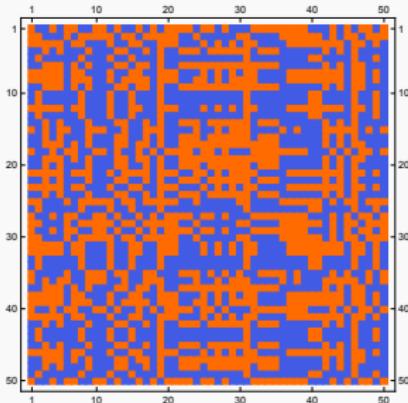
Alg. Matrix Exp. Gradient Inductive Matrix Completion with Side-Infomation
(MEG-IMCSI)

MEG-IMCSI Examples and Bounds

- Distinct applications induce distinct tunings and bounds of

$$\|\mathbf{U}\|_{\max}^2 \mathcal{D}_{\mathbf{M}, \mathbf{N}}^\gamma(\mathbf{U}) \leq \gamma^{-2} \hat{\mathcal{D}} \leq \text{"interpretable properties of } \mathbf{U} \text{ and side-kernels"}$$

- We give bounds dependent on **latent block structure (biclustering)**
- A matrix is (k, ℓ) -biclustered if after a permutation of the rows and columns the resultant is a $k \times \ell$ grid of rectangles each labeled as -1 or 1.



A $(9, 9)$ -biclustered 50×50 matrix before/after permuting into latent blocks.

- In the following we look at bounding $\|\mathbf{U}\|_{\max}$ via latent block structure.

Latent Block Models (definition)

A (k, ℓ) -biclustering is determined by discrete latent k -row (ℓ -column) states. And a $k \times \ell$ interaction matrix \mathbf{U}^* .

Definition

The set of (k, ℓ) -biclustered matrices is

$$\mathbb{B}_{k,\ell}^{m,n} := \left\{ \boldsymbol{U} \in \{\pm 1\}^{m \times n} : \begin{matrix} \boldsymbol{r} \in [k]^m \\ \boldsymbol{c} \in [\ell]^n \end{matrix}, \boldsymbol{U}^* \in \{\pm 1\}^{k \times \ell}, U_{ij} = U_{r_j c_i}^*, \forall i \in [m], j \in [n] \right\}.$$

Alternatively, define **block expansion** matrices $\mathcal{B}^{m,d}$,

$$\mathcal{B}^{m,d} := \{\mathbf{B} \in \{0,1\}^{m \times d} : \|\mathbf{B}_j\| = 1, i \in [m], \text{rank}(\mathbf{B}) = d\}$$

$$R \quad \boxed{C^T = }$$

Thus a (k, ℓ) -biclustered matrix $\textcolor{orange}{U}$ may be decomposed,

$$\mathbf{U} = \mathbf{R}\mathbf{U}^*\mathbf{C}^\top \text{ for } \mathbf{U}^* \in \{\pm 1\}^{k \times \ell}, \mathbf{R} \in \mathcal{B}^{m,k} \text{ and } \mathbf{C} \in \mathcal{B}^{n,\ell}.$$

Latent models : Low-Rank vs Small Biclustering

Rank Complexity

A matrix $\mathbf{U} \in \mathbb{R}^{m \times n}$ has a rank- d decomposition if there exists $\mathbf{P} \in \mathbb{R}^{m \times d}$ and $\mathbf{Q} \in \mathbb{R}^{n \times d}$ such that $\mathbf{U} = \mathbf{P}\mathbf{Q}^\top$. Observe that a rank- d decomposition is specified by $d(m + n)$ real parameters.

Biclustering Complexity

A matrix $\mathbf{U} \in \mathbb{R}^{m \times n}$ has a (k, ℓ) -biclustering if there exists $p \in [k]^m$, $q \in [\ell]^d$, and $\mathbf{U}^* \in \mathbb{R}^{k \times \ell}$ such that $U_{ij} = U_{p_i, q_j}^*$. Observe that a (k, ℓ) -biclustering is specified by $k\ell$ real parameters and $m + n$ integers. Define $\text{bcd}(\mathbf{U}) := \min(k, \ell)$ where (k, ℓ) is the minimal bicluster of \mathbf{U} .

Inequalities

$$\|\mathbf{U}\|_{\max}^2 \leq \text{rank}(\mathbf{U}) \leq \text{bcd}(\mathbf{U})$$

Latent Models – Summary

- Low-rank assumption each row/col is associated with latent factors in $P_i, Q_j \in \mathbb{R}^d$ and the matrix entry $U_{ij} = P_i \cdot Q_j$.
- Biclustering assumption each row/col is associated with latent factors in $p(i) \in [k], q(j) \in [\ell]$ and the matrix entry U_{ij} is an arbitrary function of $p(i)$ and $q(j)$.
- Max-norm may be much smaller than rank!

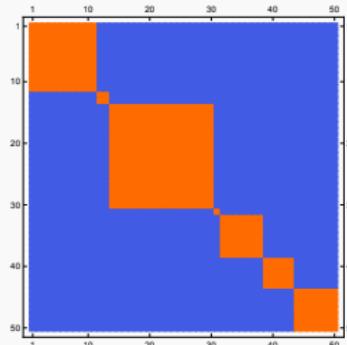
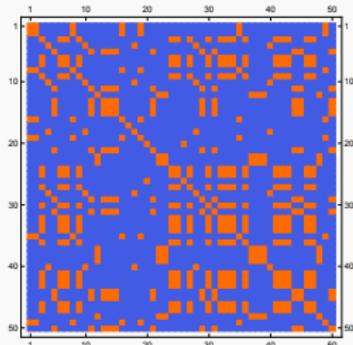
Max-norm of Similarity (Community) Prediction

Definition

The class of $m \times m$ **similarity** matrices \mathcal{S}^m is defined as,

$$\begin{aligned}\mathcal{S}_k^m &:= \{\mathbf{U} = \mathbf{R}\mathbf{S}_k^*\mathbf{C}^\top : \mathbf{R} \in \mathcal{B}^{m,k} \text{ and } \mathbf{C} \in \mathcal{B}^{m,k}\} \subset \mathbb{B}_{k,k}^{m,m} \\ \mathcal{S}^m &:= \cup_{k=1}^m \mathcal{S}_k^m\end{aligned}$$

where $(\mathbf{S}_k^*)_{ij} := (2[i == j] - 1)$; “Identity” matrix with ‘0’s \rightarrow ‘-1’s.



Before and After Permutation (\mathcal{S}_7^{50})

- Observe that if $\mathbf{U} \in \mathcal{S}_k^m$ then $\text{rank}(\mathbf{U}) = k$ but $\|\mathbf{U}\|_{\max}^2 = \mathcal{O}(1)$.

Adding Side-Information

Adding Side-Information

Side-Information

- Now assume side information about each row and each column.
- For example with movies we might have “genre” information and for the users we might have “demographics.”
- This side information is expressed via a kernel on the rows $\mathcal{M}^+ : \mathcal{I} \times \mathcal{I} \rightarrow \mathbb{R}$ and a kernel on the columns $\mathcal{N}^+ : \mathcal{J} \times \mathcal{J} \rightarrow \mathbb{R}$.
- The case of no side-information is supplying “identity matrices” for which we have

$$\mathcal{D}_{I_m, I_n}^\gamma(\mathbf{U}) = m + n.$$

Bounding the quasi-dimension \mathcal{D} (Theorem)

Theorem ([A,HPT20a], [B,HPT20b])

[A]: If $\mathbf{U} \in \mathbb{B}_{k,\ell}^{m,n}$ then define

$$\mathcal{D}_{\mathbf{M},\mathbf{N}}^{\circ}(\mathbf{U}) := \begin{cases} 2 \operatorname{tr}(\mathbf{R}^{\top} \mathbf{MR}) \mathcal{R}_{\mathbf{M}} + 2 \operatorname{tr}(\mathbf{C}^{\top} \mathbf{NC}) \mathcal{R}_{\mathbf{N}} + 2k + 2\ell & (C1) \\ k \operatorname{Tr}(\mathbf{R}^{\top} \mathbf{MR}) \mathcal{R}_{\mathbf{M}} + \ell \operatorname{Tr}(\mathbf{C}^{\top} \mathbf{NC}) \mathcal{R}_{\mathbf{N}} & (C2) \end{cases}.$$

as the minimum over all decompositions of $\mathbf{U} = \mathbf{RU}^* \mathbf{C}^{\top}$ for $\mathbf{R} \in \mathcal{B}^{m,k}$, $\mathbf{C} \in \mathcal{B}^{n,\ell}$ and $\mathbf{U}^* \in \{-1, 1\}^{k \times \ell}$ then,

$$\mathcal{D}_{\mathbf{M},\mathbf{N}}^{\gamma}(\mathbf{U}) \leq \mathcal{D}_{\mathbf{M},\mathbf{N}}^{\circ}(\mathbf{U}) \quad (\text{if } \|\mathbf{U}\|_{\max} \leq 1/\gamma)$$

(C1) : \mathbf{M} and \mathbf{N} are “PDLaplacians”

(C2) : \mathbf{M} and \mathbf{N} are strictly positive definite.

[B]: If $\mathbf{U} \in \mathbb{B}_{m,\ell}^{m,n}$, $\gamma = 1/\sqrt{\ell}$, and (C2) holds then define

$$\mathcal{D}_{\mathbf{M},\mathbf{N}}^*(\mathbf{U}) := \gamma^2 \operatorname{tr}((\mathbf{U}^*)^{\top} \mathbf{MU}^*) \mathcal{R}_{\mathbf{M}} + \operatorname{tr}(\mathbf{C}^{\top} \mathbf{NC}) \mathcal{R}_{\mathbf{N}}$$

as min. over decomp. of $\mathbf{U} = \mathbf{U}^* \mathbf{C}^{\top}$ for $\mathbf{U}^* \in \{-1, 1\}^{m \times \ell}$ and $\mathbf{C} \in \mathcal{B}^{n,\ell}$ then

$$\mathcal{D}_{\mathbf{M},\mathbf{N}}^{\gamma}(\mathbf{U}) \leq \mathcal{D}_{\mathbf{M},\mathbf{N}}^*(\mathbf{U}) \quad (\gamma = 1/\sqrt{\ell}).$$

Bounding the quasi-dimension \mathcal{D} (Intuitions)

Intuitions

- We've replaced the row-normalised matrices $\hat{\mathbf{P}}^\top$ ($\hat{\mathbf{Q}}^\top$) in \mathcal{D} with the "one-hot" row-normalised matrices \mathbf{R}^\top (\mathbf{C}^\top).
- The term $\text{tr}(\mathbf{R}^\top \mathbf{M} \mathbf{R}) \mathcal{R}_M$ closely resembles the mistake bound for the multi-class kernel perceptron. Where the classes are the "latent factors"
- Qualitatively $\text{tr}(\mathbf{R}^\top \mathbf{M} \mathbf{R}) \mathcal{R}_M$ will be small if latent row factors are "well-predicted" via kernel \mathcal{M}^+ .
- The inequality [A] in its upper bound depends on \mathbf{U}^* only through its dimensionality and the fact that $\mathbf{U} = \mathbf{R} \mathbf{U}^* \mathbf{C}^\top$.
- Inequality [A] is independent of γ except for requirement $\|\mathbf{U}\|_{\max} \leq 1/\gamma$
- Inequality [B] depends on γ to be its "worst-case" $\|\mathbf{U}\|_{\max} = 1/\gamma$
- For the results in this talk we need only inequality [B]

The Reduction

Reducing to Inductive Matrix Completion with Side-Information

1. The column space corresponds to *time*.
2. The row space corresponds to the input space of kernel K .
3. The “time” kernel P engenders the bias that adjacent columns should be similar.

The Graph Laplacian

Definition

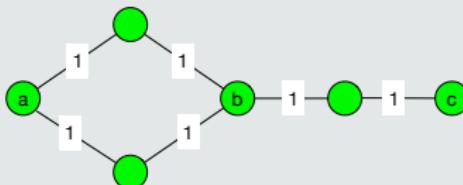
The graph Laplacian is $L(G) := D - W$ where W is the (symmetric non-negatively weighted) adjacency matrix and $D := \text{diag}(d_1, \dots, d_n)$ is diagonal matrix of (weighted) degrees where $d_v := \sum_{i \neq v} w_{vi}$.

Properties

- The laplacian $L(G)$ is positive semi-definite.
- Observe

$$\mathbf{u}^\top L \mathbf{u} := \sum_{(i,j) \in E(G)} w_{ij} (u_i - u_j)^2$$

- Define $r(i,j)$ to be the effective resistance between vertex i and j if we treat G as a resistive circuit with each edge (resistor) connected with inherent resistance $\frac{1}{w_{ij}}$.



Resistive Network with unit resistors with $r(a,b) = 1$ and $r(b,c) = 2$.

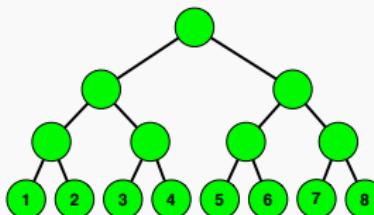
- The diagonal of the “kernel” is bounded by the (effective) resistance diameter

$$\mathcal{R}_L := \max_i L_{ii}^+ \leq \max_{i,j} r(i,j)$$

Path-Tree Kernel

Path-Tree Kernel $P(\cdot, \cdot)$

A *path-tree kernel* $P : [T] \times [T] \rightarrow \mathbb{R}$, is formed via the Laplacian of a fully complete binary tree with $N := 2^{\lceil \log_2 T \rceil + 1} - 1$ vertices. The *path* corresponds to the first T leaves of the tree.



Path-Tree Kernel (Length 8)

Lemma (See [HLP09, HPT20b])

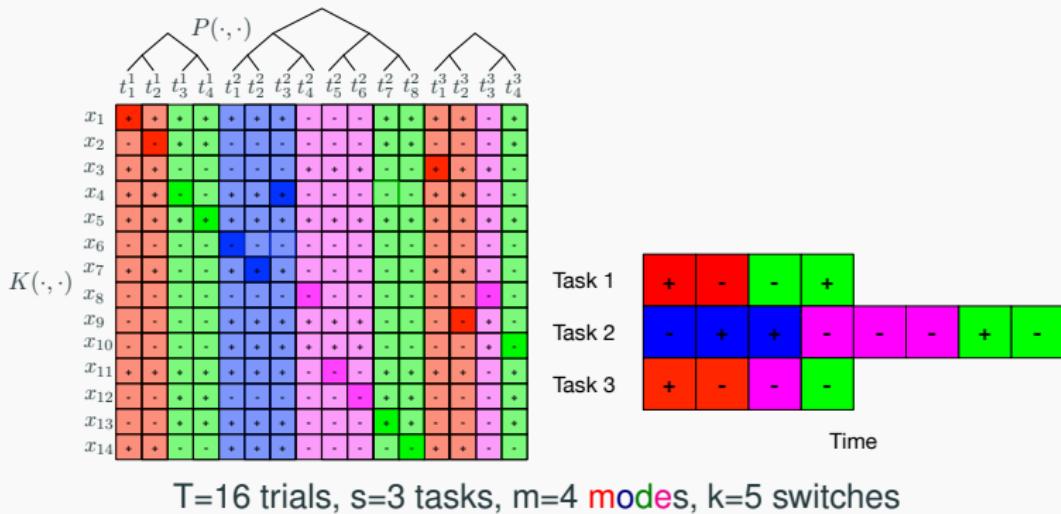
If $f \in \mathcal{H}_P \cap \{0, 1\}^T$ then

$$\max_{t \in [T]} P(t, t) \leq 2^{\lceil \log_2 T \rceil}$$

$$\|f\|_P^2 \max_{t \in [T]} P(t, t) \leq k(f) \lceil \log_2 T \rceil^2 + 2, ,$$

where $k(f) := \sum_{t=1}^{T-1} [f(t) \neq f(t+1)]$.

Illustration



$T=16$ trials, $s=3$ tasks, $m=4$ modes, $k=5$ switches

- Emphasised matrix entries correspond to observations.
- Sparse matrix completion at most one observation per column.

Proof Sketch

Theorem

$$R \leq \mathcal{O} \left(\sqrt{\left(\sum_{i=1}^m \|h_i\|_K^2 X^2 + km(\log T)^2 \right) T \log T} \right)$$

Proof Sketch

1. Recall

$$\sum_{t \in [T]} \mathbb{E}[\mathcal{L}_{01}(y_t, \hat{y}_t)] - \mathcal{L}_{01}(y_t, U_{ij_t}) \leq 4 \sqrt{2 \frac{\hat{\mathcal{D}}}{\gamma^2} \log(2T) T}$$

for all $\mathbf{U} \in \{-1, 1\}^{m \times n}$ with $\|\mathbf{U}\|_{\max} \leq 1/\gamma$.

2. Observe that $\mathbf{U} \in \mathbb{B}_{T,m}^{T,T}$ (See “Illustration”) thus $\|\mathbf{U}\|_{\max} \leq \sqrt{m}$.

3. Using

$$\mathcal{D}_{\mathbf{M}, \mathbf{N}}^*(\mathbf{U}) := \gamma^2 \text{tr}((\mathbf{U}^*)^\top \mathbf{M} \mathbf{U}^*) \mathcal{R}_{\mathbf{M}} + \text{tr}(\mathbf{C}^\top \mathbf{N} \mathbf{C}) \mathcal{R}_{\mathbf{N}}$$

4. Thus,

$$\hat{\mathcal{D}} \gamma^{-2} = \sum_{i=1}^m \|h_i\|_K^2 X^2 + m \text{tr}(\mathbf{C}^\top P^{-1} \mathbf{C}) \mathcal{R}_{\mathbf{N}}$$

5. By Lemma (assume $s \leq k$) we have $\text{tr}(\mathbf{C}^\top P^{-1} \mathbf{C}) \in k(\log T)^2$. ■.

Open Problems

1. Improve algorithm efficiency for RKHS $O(T^3)$.
2. Improve RKHS algorithm to replace $m \rightarrow \log m$.
3. Lower Bounds (however, see [HPT20b, Prop. 4]).
4. Self-tuning for the parameters.
5. Alternate hypothesis classes (beyond RKHS and finite hypotheses classes).
6. Loss functions beyond 0 – 1.
7. Incorporate “drift” into switching model.
8. Algorithms considered are centralized. Distributed algs. with limited communication possible?
9. Experiments!

Thanks



References – 1

- [LW94] N. Littlestone and M. Warmuth. *The weighted majority algorithm*, (1994)
- [FS97] Y. Freund and R. Schapire. *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*, (1997).
- [V90] V. Vovk, *Aggregating strategies*, (1990).
- [CLW96] N. Cesa-Bianchi, P. Long, and M. Warmuth. *Worst-case Quadratic Loss Bound for Prediction Using Linear Functions and Gradient Descent*, (1996)
- [S11] S. Shalev-Schwartz, *Online Learning and Online Convex Optimization*, (2011)
- [HW98] M. Herbster and M. Warmuth. *Tracking the Best Expert*, (1998)
- [F00] Y. Freund. *Private communication*, (2000). Also posted on <http://www.learning-theory.org>
- [BW03] O. Bousquet and M.K. Warmuth. Tracking a small set of experts by mixing past posteriors.
- [KAW12] W. Koolen, D. Adamskiy, and M. Warmuth. *Putting Bayes to Sleep*, (2012)
- [HPT20b] M. Herbster, S. Pasteris and L. Tse. *Online Multitask Learning with Long-Term Memory*, (2020).

References – 2

- [HPT20a] M. Herbster, S. Pasteris, and L. Tse. *Online matrix completion with side information*, (2020).
- [LMSS07] N. Linial, S. Mendelson, G. Schechtman, and A. Shraibman. *Complexity measures of sign matrices*, (2007).
- [TRW05] K. Tsuda, G. Ratsch, and M.K. Warmuth. *Matrix exponentiated gradient updates for on-line learning and bregman projection*, (2005).
- [SSSHZ15] S. Sabato, S. Shalev-Shwartz, N. Srebro, Daniel J. Hsu, and T. Zhang. *Learning sparse low-threshold linear classifiers*, (2015).
- [GHP13] C. Gentile, M. Herbster, and S. Pasteris. *Online similarity prediction of networked data from known and unknown graphs*, (2013).
- [HKS12] E. Hazan, S. Kale, and S. Shalev-Shwartz. *Near-optimal algorithms for online matrix prediction*, (2013).
- [HP09] M. Herbster, G. Lever, and M. Pontil. *Online prediction on large diameter graphs* (2009)