VII Efficient Learning II – A Hardness Result.

1. Introduction and Review
2. Learning Conjunctions of Clauses C_n^k is Hard
3. Summary of Lecture

1 Introduction and Review

We have asked:

- Can algorithm L PAC learn in hypothesis space H?
- Can L PAC learn in H *efficiently*?

Now we ask:

- Is there *any* efficient algorithm for learning in H?

We will see there is probably no efficient algorithm for $H = C = C_n^k$.

Hard and Easy Problems

From last lecture:

- Easy problems are those with polynomial time algorithms.
- Hard problems are those without polynomial time algorithms.

To show a problem is easy: give a polynomial algorithm for it.

But to show a problem is hard: we have to show there is no polynomial algorithm for it.

How can we do this? Relate the problem to a known hard problem.

How to Show a Problem is Hard

Suppose:

- We want to show that Π is a hard problem.
- We know another problem Π_0 is NP-Complete (i.e. hard).

Suppose we can show that if there is a polynomial time algorithm for Π there must also be one for Π_0.

If so, Π is NP-hard: if the $\mathrm{P} \neq \mathrm{NP}$ conjecture turns out to be true, then there is no polynomial time algorithm for Π.

2 Learning Conjunctions of Clauses C_n^k is Hard

Let's show that learning conjunctions of clauses with $H = C$ is hard.

- I.e. that there is (probably) no efficient algorithm for this H.
- We say *probably* because the $\mathrm{P} \neq \mathrm{NP}$ conjecture has not been proved, although it is widely believed.

Note that we assume the hypothesis space H is the same as the concept space C.

Next week we will see what happens when $H \neq C$.

Clauses and C_n^k

A *clause* is a disjunction of literals, e.g.

$$u_2 \vee \bar{u}_3 \vee u_5$$

C_n is the space of clauses on n variables.

C_n^k is the space of conjunctions (ANDs) of k clauses on n variables, e.g.

$$(u_1 \vee \bar{u}_2) \ \wedge \ (\bar{u}_1 \vee u_3) \ \wedge \ (u_3 \vee u_4)$$

is a conjunction of $k = 3$ clauses, and so belongs to C^3.

Search and Existence Problems

Search problems:

- What is the biggest number in $\{3, 9, 0, 5, -1\}$?
- What hypotheses are consistent with the training sequence?

Existence problems:

- Is there a positive number in $\{3, 9, 0, 5, -1\}$?
- Is there any hypothesis in H consistent with the training sequence?

Search is Harder than Existence

Existence problems are easier: if we solve a search problem we have solved the corresponding existence problem.

- If we know the biggest number in a list is 9, we know there exists a positive number.

The reverse is not true:

- If we know there exists a positive number in the list, we do *not* know what the biggest number is.

H-Consistency

Learning a hypothesis which approximates a target concept is a search problem:

- What is a good hypothesis?

Learning a hypothesis is harder than the corresponding existence problem:

- Is there a hypothesis in H which is consistent with the training sequence?
  - This is called the *H-consistency* problem.
  - We are interested in learning in $H = C_n^k$ so we will look at the C_n^k-consistency problem.

Showing Learning is Hard by Showing H-Consistency is Hard

If
H-consistency (an existence problem) is hard
then
learning (a search problem) must be hard
because
H-consistency is easier than learning.

So: if we can show that H-consistency is hard, we know that learning is hard.

Proof Outline

We'll show that learning in $H = C_n^k$ is hard by showing that $H = C_n^k$-consistency is hard.

This is how:

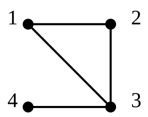
- Graph k-colouring is known to be a hard problem.
- We'll relate k-colouring to C_n^k-consistency.
- It follows that C_n^k-consistency is also hard.
- Since C_n^k-consistency is easier than learning in $H = C_n^k$, they must both be hard.

Graphs

A *graph* is a set of *vertices* $V = \{1, \ldots n\}$ and *edges* E.

An edge connects two vertices and is represented by them: $E \subseteq V \times V$.

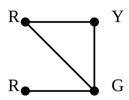
For example, suppose $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (2, 3), (3, 4)\}$. The graph $G = (V, E)$ is:



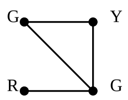
Graph k-colouring

A *k-colouring* of a graph is the assignment of one of k colours to each vertex, so that no edge has the same colour at both ends.

Here are two graphs with $k = 3$ colours (R=Red, Y=Yellow, G=Green):



A k-colouring



Not a k-colouring

The existence problem for k-colourings is known to be NP-complete (i.e., hard) for $k \geq 3$ (Garey and Johnson 1979).

Relating Graph k-colouring and C_n^k-consistency

We want to relate two problems: graph k-colouring and C_n^k-consistency.

How can we do this?

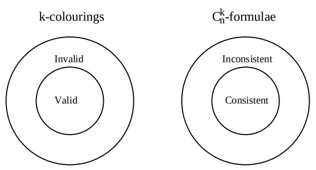
Suppose we want to show two sets A and B are equal.

If all elements of A are in B **and** all elements of B are in A **then** A and B are equal	in other words	**If** $\forall a \in A,\ a \in B$ **and** $\forall b \in B,\ b \in A$ **then** A = B

Relating Graph k-colouring and C_n^k-consistency

We'll use two sets: the set of valid k-colourings, and the set of consistent formulae in C_n^k.

We'll show that every element in one set corresponds to an element of the other set.



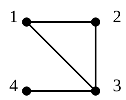
Relating Graph k-colouring and C_n^k-consistency

To relate the two problems we transform k-colouring problems into C_n^k-consistency problems.

To do this we construct a training sequence s(G) from graph G as follows:

- For each vertex $i \in V$ we take as a negative example the vector v_i which has a 1 in the ith position and 0 elsewhere.
- For each edge $ij \in E$ we take as a positive example the vector $v_i + v_j$.

G



s(G)

Negative	Positive
(1000,0)	(1100,1)
(0100,0)	(1010,1)
(0010,0)	(0110,1)
(0001,0)	(0011,1)

Proposition 7.1 There is a function in C_n^k which is consistent with the training sequence $\mathbf{s}(G)$ if and only if graph G is k-colourable.

Proof outline:

- All valid k-colourings are consistent with some formula in C_n^k.
- All consistent formulae in C_n^k define a valid k-colouring.

In other words, we will show:

- All elements of A correspond to an element of B.
- All elements of B correspond to an element of A.

Proposition 7.2 All consistent formulae in C_n^k define a valid k-colouring.

Proof

We must do two things:

1. Give a method of converting a formula to a colouring.
2. Show that this always produces valid colourings.

From Formula to Colouring

Here is an algorithm to convert from a formula to a colouring.

Assign each clause a colour. E.g., for $k = 3$:

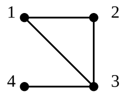
$$h = \underbrace{h_1}_{Red} \wedge \underbrace{h_2}_{Yellow} \wedge \underbrace{h_{k=3}}_{Green}$$

1. For each vertex $v \in V$
2. Find the first clause in $h = h_1 \wedge h_2 \wedge \ldots \wedge h_k$ which is consistent with v_i (the example for vertex i).
3. The colour of the vertex is the colour of that clause.

Example

Find the colouring of G

G



s(G)

Negative	Positive
(1000,0)	(1100,1)
(0100,0)	(1010,1)
(0010,0)	(0110,1)
(0001,0)	(0011,1)

defined by the formula

$$h = \langle\underbrace{(u_2 \vee u_3 \vee u_4)}_{Green} \wedge \underbrace{(u_1 \vee u_3)}_{Red} \wedge \underbrace{(u_1 \vee u_2 \vee u_4)}_{Yellow}\rangle$$

Hint: Vertex 1 is Green because (1000,0) is negative and does not satisfy $(u_2 \vee u_3 \vee u_4)$.

Does this Method Give a Colouring?

Is it possible that a vertex does not get a colour?

Suppose we have a hypothesis $h \in C_n^k$ which is consistent with $\mathbf{s}(G)$. By definition, h is conjunction of clauses:

$$h = h_1 \wedge h_2 \wedge \ldots \wedge h_k$$

The examples in $\mathbf{s}(G)$ derived from vertices are negative.

Since h is consistent with $\mathbf{s}(G)$, it outputs 0 (negative) for each example v_i from a vertex i of G. That is, $h(v_i) = 0$.

Since $h(v_i) = 0$, at least one of the clauses in h must be 0. (If each clause $h_1 \wedge h_2 \wedge \ldots \wedge h_k$ was 1, then h would be 1 also.)

In other words, each vertex's example must satisfy at least 1 clause, and so each vertex gets a colour.

From Formula to Colouring – Alternative Method

Anthony and Biggs specify a function χ which converts from a consistent formula to a colouring:

$$\chi(i) = min\{f \mid h_f(v_i) = 0\}$$

where i is a vertex, f indexes clauses and h_f is a clause in h.

This method uses numbers to identify colours.

The colour of a vertex i is the index of the first clause which satisfies its example.

Is this a Valid Colouring?

We have given two equivalent methods of producing a colouring from a consistent formula.

It remains to show that any such colouring is valid (i.e., no vertices joined by an edge have the same colour).

The proof is in Anthony and Biggs, page 46. □

From Colouring to Formula

Proposition 7.3 All valid k-colourings are consistent with some formula in C_n^k.

Proof We need a way of converting any colouring to a formula:

Suppose we are given a colouring $\chi : V \to \{1, 2, \ldots, k\}$. For $1 \leq f \leq k$, define h_f to be the clause

$$\left\langle \bigvee_{\chi(i) \neq f} u_i \right\rangle$$

and define $h = h_1 \wedge h_2 \wedge \ldots \wedge h_k$.

Anthony and Biggs (page 47) contains a proof that h is consistent with $\mathbf{s}(G)$. □

We repeat proposition 7.1:

Proposition 7.1 There is a function in C_n^k which is consistent with the training sequence $\mathbf{s}(G)$ if and only if graph G is k-colourable.

Proof

- All valid k-colourings are consistent with some formula in C_n^k.
  - This is proposition 7.2, for which there is a proof in Anthony and Biggs.
- All consistent formulae in C_n^k define a valid k-colouring.
  - This is proposition 7.3, for which there is a proof in Anthony and Biggs.

□

It is now easy to show that C_n^k-consistency is NP-hard.

C_n^k-consistency is NP-hard

Assume we have a polynomial algorithm for C_n^k-consistency. We call this an 'oracle'.

Given a graph k-colouring problem we can construct a C_n^k-consistency problem in polynomial time (as shown on slide 24).

The oracle's yes/no answer to the consistency problem is the same as the yes/no answer to the k-colouring problem.

We have used the oracle to solve k-colouring in polynomial time.

I.e., the existence of a polynomial algorithm for C_n^k-consistency implies the existence of a polynomial algorithm for k-colouring.

But we know (Garey and Johnson 1979) there is no polynomial algorithm for k-colouring (unless $\mathrm{P} = \mathrm{NP}$), so there cannot be a polynomial algorithm for C_n^k-consistency.

Proof that Learning in $H = C_n^k$ is Hard

We have now related k-colouring and C_n^k-consistency.

We can now treat the proof outline from slide 11 as a proof:

Proposition 7.4 C_n^k-consistency is NP-hard.

Proof

- Graph k-colouring is known to be a hard problem.
- We have related k-colouring to C_n^k-consistency.
- It follows that C_n^k-consistency is also hard.
- Since C_n^k-consistency is easier than learning in $H = C_n^k$, they must both be hard.

□

Notes on Learning in $H = C = C_n^k$

k-colouring is NP-hard only for $k \geq 3$.

- So relating k-colouring and H-consistency proves that C_n^k-consistency is hard only for $k \geq 3$.

To show C_n^k-consistency is hard for $k = 2$ we can relate it to another NP-hard problem called *set-splitting*.

With $k = 1$, C_n^k-consistency is *not* hard; there is an efficient algorithm for it.

3 Summary

- Search and existence problems.
  - Search problems are harder.
- C_n^k formulae: conjunctions of k clauses of n variables.
- H-consistency: is there a function in H consistent with the training sequence?
- Graph k-colouring: is there a way of assigning k colours to a graph so that no two adjacent vertices have the same colour?

Summary Continued

- We saw the outline of how to relate C_n^k-consistency and k-colouring.
  - This approach can be used to prove C_n^k-consistency and learning in $H = C_n^k$ are hard.
  - In other words, there is *probably* no efficient algorithm for learning in C_n^k (i.e., unless $\mathrm{P} = \mathrm{NP}$).
- Next week we will see what happens when we learn target concepts from C_n^k, but search for hypotheses in another concept space, i.e., when $C \neq H$.

References

M. Garey and D. Johnson (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco.