Dealing with feature explosion

Common situation:

- Lots of features, but you suspect there is some small set that's good enough.
- Want performance to not degrade badly with addition of new features.

Lots of approaches studied.

Two today: Weighted-Majority, Winnow

(natural, do pretty well, can analyze)

Scenario 1: Have bunch of "experts" giving advice. What's a good way to decide what to do?

local TV	channel 38	newspaper	window	arthritis	actual
rain	rain	rain	no rain	no rain	no rain
no rain	rain	no rain	rain	rain	rain
no rain	no rain	no rain	rain	no rain	no rain
rain	no rain	rain	rain	no rain	rain

Batch method: take large sample, look at performance of each one, pick best.

What about more incremental method? Don't want to get soaked a month waiting.

What if things change?

Weighted Majority method

- Give weight to each predictor (say, start with 1).
- Predict by weighted majority vote.
- Penalize each one that gave wrong answer: Multiply by penalty β (say, 0.5).

Nice features:

- Easy to do (even if each predictor is itself a learning algorithm).
- Provably never much worse than best predictor.
- Bayesian interpretation.

Analysis:

n predictors. We make M mistakes, best predictor makes m mistakes.

- Each time we make mistake:
  (new total weight) $\leq 3/4$ (old total weight)
- At end, total weight $\leq n(3/4)^M$.
- At end, weight on best predictor $= (1/2)^m$.

Solve: $n(3/4)^M \geq (1/2)^m$.

Get: $M \leq 2.4(m + \log_2 n)$.

- What can you do if you expect best predictor to change over time?

Example:

On calendar data, suppose you think there is some pair of features that makes a good predictor.

Simple predictor for feature pair (f_1, f_2):

- Given a new example with $f_1 = x, f_2 = y$, look at last few (say 5) times f_1 was x and f_2 was y and take majority or most recent on ties.

Create one algorithm for each pair and run WM.

Results: 528 algs (ignore 2 features with list-valued attrs). Makes 6 mistakes on test. Also, after 1st 10 training examples, makes 16 more mistakes on remaining 123 training data pts.

In practice can throw out predictors that falter badly. Get algorithm that speeds up as it learns.

```
Top pair is (req-start-time, group-attendees?)
next are (req-start-time,cmu-attendees) and (req-start-time,known attendees
End result: 6 mistakes on test set.

After training set, top rule is:
situation: 1500,unknown   last 4 seen: no no no no
situation: 1630,no        last 4 seen: yes yes yes yes
situation: 1530,unknown   last 4 seen: no no no no
situation: 1330,no        last 4 seen: yes yes yes yes
situation: 1400,no        last 4 seen: yes no yes yes
situation: 1430,no        last 4 seen: yes yes yes yes
situation: 1200,unknown   last 4 seen: no no no no
situation: 1500,no        last 4 seen: yes yes yes yes
situation: 830,yes        last 4 seen: no no no no
situation: 1700,no        last 1 seen: no
situation: 1200,yes       last 4 seen: no no no no
situation: 1130,unknown   last 2 seen: yes yes
situation: 1630,yes       last 4 seen: no no no no
situation: 1500,yes       last 4 seen: no no no no
situation: 1330,yes       last 1 seen: yes
situation: 1700,yes       last 2 seen: no no
situation: 800,yes        last 3 seen: no no no
situation: 800,unknown    last 3 seen: no no no
situation: 900,unknown    last 2 seen: no no
situation: 1130,no        last 2 seen: no no
situation: 1100,no        last 1 seen: yes
situation: 1730,unknown   last 1 seen: yes
situation: 1200,no        last 2 seen: no no
situation: 1000,yes       last 1 seen: yes
situation: 1300,no        last 1 seen: yes
situation: 1600,yes       last 1 seen: yes
situation: 1230,no        last 3 seen: no no no
situation: 1600,unknown   last 1 seen: no
situation: 1530,yes       last 1 seen: no
situation: 1030,no        last 1 seen: yes
situation: 900,yes        last 1 seen: no
situation: 900,no         last 1 seen: no
```

Scenario 2: Have n features. Target is disjunction of r of them. Don't know which r.

E.g., *doesn't-enjoy-sport:* `cold` OR `rainy`.

E.g, Calendar: one "feature" for each rule of size ≤ 2. Try to find best disjunction.

Goal 1: If r is small, n is large, do well.

Goal 2: be robust to noise.

Winnow algorithm

- Use linear threshold function for prediction.

  predict positive when $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta$.

- Start: all $w_i = 1$, threshold $= n$, say.

- Update rule: For each example, compare prediction to correct value.
  - Mistake on positive:

    If $x_i = 1$ then $w_i \leftarrow 2w_i$.
  - Mistake on negative:

    If $x_i = 1$ then $w_i \leftarrow w_i/2$.

Example:

011011 +
100101 −
101110 +
010001 +
101001 −

Analyze in mistake-bound model: assume no noise.

Lets call weights on features in target the "relevant weights."

- Relevant weights never decrease.
- Each mistake on positive doubles at least one relevant weight.
- Each relevant weight can be doubled at most $\log_2 n + 1$ times.

So, at most $r(\log_2 n + 1)$ mistakes on positives.

Now look at total weight:

- Total weight starts at n.
- Each mistake on positive adds at most n to total weight.
- Each mistake on negative removes at least $n/2$ from total weight.

Total weight always > 0.

$$\begin{aligned}(\#\text{ mistakes on neg}) &< \frac{n + n(\#\text{ mistakes on pos})}{n/2} \\ &< 2 + 2r(\log_2 n + 1)\end{aligned}$$

So, total number of mistakes at most $2 + 3r(\log_2 n + 1)$.

How do things break down when even the best disjunction makes mistake?

- Negative with t features in best disjunction: decreases t relevant weights.
- Positive with 0 features in best disjunction: adds up to n to total weight.

Results on calendar data:

After two passes through training, 62 correct and 5 mistakes on test.

WM/winnow hybrid (not theoretically justified): 64 correct and 3 mistakes with 1 pass, 65 correct and 2 mistakes with 2 passes.