

Margins, Mistakes, Ranking, Sequential Learning and the Perceptron

William Cohen

October 4, 2009

1 Online mistake-bounded learning for classification

Bayesian probability is the most-studied mathematical model of learning (here at CMU, anyway). But there are other models that also are experimentally successful, and give useful insight. Sometimes these models are also mathematically more appropriate - e.g., a function that models $P(Y|X)$ may not be the best function to use if you want to maximize accuracy.

Another useful model is this. Consider an *on-line learner* L . L has two methods:

- $L.predict(\mathbf{x})$ returns some real number \hat{y} . The sign of \hat{y} is a binary prediction, “positive” or “negative”.
- $L.addExample(\mathbf{x}, y)$ is called to “tell” L that y is the correct label for \mathbf{x} . Here $y \in -1, +1$.

Notation and stuff: \mathbf{x} and \mathbf{u} are vectors, y is a real number, $\mathbf{x} \cdot \mathbf{u}$ is dot product, and $\|\mathbf{x}\| = \sqrt{\mathbf{x} \cdot \mathbf{x}}$. As some background, remember that if θ is the angle between \mathbf{x} and \mathbf{u} ,

$$\cos \theta = \frac{\mathbf{x} \cdot \mathbf{u}}{\|\mathbf{x}\| \|\mathbf{u}\|}$$

This is important if you’re trying to keep a picture in your mind of what all these dot-products mean. A special case is that $\mathbf{x} \cdot \mathbf{u} = \|\mathbf{x}\| \cos \theta$ when

$\|\mathbf{u}\| = 1$. Visually, for a unit vector \mathbf{u} , $\mathbf{x} \cdot \mathbf{u}$ is the distance you get “if you project \mathbf{x} in the direction \mathbf{u} ”.

Now consider this game, played between two players A and B . Player A provides examples $\mathbf{x}_1, \dots, \mathbf{x}_T$ for each round (chosen arbitrarily).

In each round, B first makes a prediction \hat{y}_i (say, using a learner L). Then A picks a label y_i , arbitrarily, to assign to \mathbf{x}_i . If $\text{sign}(y_i) \neq \text{sign}(\hat{y}_i)$...or if you prefer, if $y_i \hat{y}_i < 0$...then B has made a *mistake*. A is trying to force B to make as many mistakes as possible.

To make this reasonable, we need a few more constraints on what A is allowed to do, and what B will do. Each set of constraints gives you a new theorem (maybe, if you can work it out!).

1.1 The perceptron game

Here’s the version of the online-learning game we’ll consider today. There are three extra rules, to make the game “fair” for B .

Margin γ . A must provide examples that can be separated with some vector \mathbf{u} with margin $\gamma > 0$, ie

$$\exists \mathbf{u} : \forall (\mathbf{x}_i, y_i) \text{ given by } A, (\mathbf{u} \cdot \mathbf{x}_i)y_i > \gamma$$

and furthermore, $\|\mathbf{u}\| = 1$.

To make sense of this, recall that if θ is the angle between \mathbf{x} and \mathbf{u} , then

$$\cos \theta = \frac{\mathbf{x} \cdot \mathbf{u}}{\|\mathbf{x}\| \|\mathbf{u}\|}$$

so if $\|\mathbf{u}\| = 1$ then $\|\mathbf{x}\| \cos \theta = \mathbf{x} \cdot \mathbf{u}$. In other words, $\mathbf{x} \cdot \mathbf{u}$ is exactly the result of projecting \mathbf{x} onto vector \mathbf{u} , and $\mathbf{x} \cdot \mathbf{u} > \gamma$ means that \mathbf{x} is distance γ away from the hyperplane \mathbf{h} that is perpendicular to \mathbf{u} . This hyperplane \mathbf{h} is the separating hyperplane. Notice that $y(\mathbf{x} \cdot \mathbf{u}) > \gamma$ means that \mathbf{x} is distance γ away from \mathbf{h} on the “correct” side of \mathbf{h} .

Radius R . A must provide examples “near the origin”, ie

$$\forall \mathbf{x}_i \text{ given by } A, \|\mathbf{x}_i\|^2 < R$$

B ’s strategy. B uses this learning strategy.

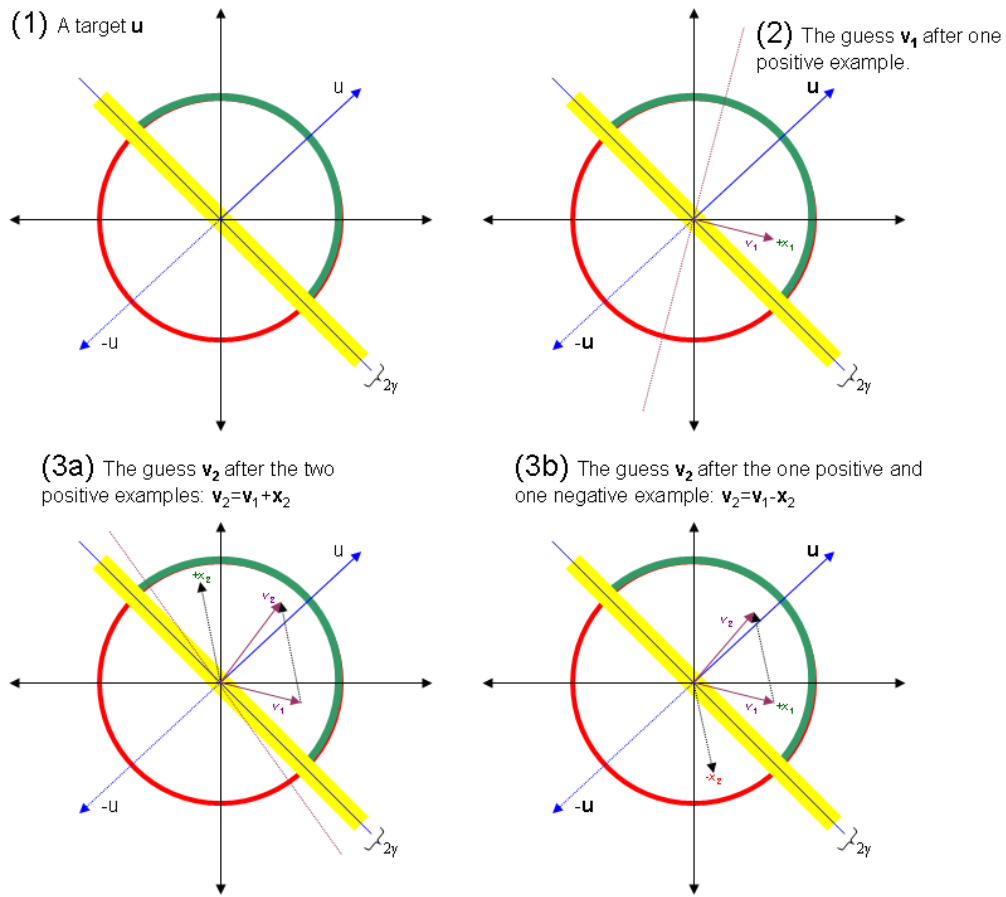


Figure 1: A couple of rounds of the perceptron game, played in two dimensions.

1. B 's initial guess is $\mathbf{v}_0 = \mathbf{0}$.
2. At every round, B 's prediction is $\mathbf{v}_k \cdot \mathbf{x}_i$, where \mathbf{v}_k is the current guess.
3. When B makes a mistake (i.e., $(\mathbf{v}_k \cdot \mathbf{x}_i)y_i < 0$) then B updates the guess as follows:

$$\mathbf{v}_{k+1} = \mathbf{v}_k + y_i \mathbf{x}_i$$

1.2 Some exercises

Don't turn these in, but do try this at home:

- Play the perceptron game with yourself or a friend in two dimensions on paper or a whiteboard. Start out by drawing \mathbf{u} and the corresponding hyperplane, the separating margin γ , and the radius R . Then draw each \mathbf{v}_k . After each mistake, measure $\mathbf{v}_k \cdot \mathbf{u}$ visually by projecting \mathbf{v}_k onto \mathbf{u} .
- Suppose that you were A and could either make \mathbf{u} large, or make R large. How could you force B to make lots and lots of mistakes?

1.3 The mistake bound

If you do follow the rules, this simple algorithm is simple to analyze. Amazingly this doesn't depend at all on the dimension of the \mathbf{x} 's, and the proof is very simple.

Theorem 1 *Under these rules, it is always the case that $k < R/\gamma^2$. In other words, the number of B 's mistakes is bounded by R , the radius the examples fall into, and $1/\gamma^2$, the square of the inverse of the margin.*

The trick is to show two things: that $\mathbf{v}_k \cdot \mathbf{u}$ grows, but $\|\mathbf{v}_k\|$ stays small.

Lemma 1 $\forall k, \mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$. *In other words, the dot product between \mathbf{v}_k and \mathbf{u} increases with each mistake, at a rate depending on the margin γ .*

Proof:

$$\begin{aligned}\mathbf{v}_{k+1} \cdot \mathbf{u} &= (\mathbf{v}_k + y_i \mathbf{x}_i) \cdot \mathbf{u} \\ \Rightarrow \mathbf{v}_{k+1} \cdot \mathbf{u} &= (\mathbf{v}_k \cdot \mathbf{u}) + y_i (\mathbf{x}_i \cdot \mathbf{u}) \\ \Rightarrow \mathbf{v}_{k+1} \cdot \mathbf{u} &\geq \mathbf{v}_k \cdot \mathbf{u} + \gamma \\ \Rightarrow \mathbf{v}_k \cdot \mathbf{u} &\geq k\gamma\end{aligned}$$

The first step is natural to consider—we're just taking the definition of \mathbf{v}_{k+1} , and since we care about the dot product with \mathbf{u} , we dot-product both sides with \mathbf{u} and then grind away. These steps follow from “ B 's strategy”; distributivity; the “margin γ ” assumption; and then induction; respectively. That's the first lemma.

Now, $\mathbf{v}_k \cdot \mathbf{u}$ could grow even if \mathbf{v}_k points “away from” \mathbf{u} —i.e., the angle between them stays large—if $\|\mathbf{v}_k\|$ grows as well. Again, let’s start with the definition of \mathbf{v}_{k+1} and “square” each side (in the dot-product sense) to see what happens to the norm.

Lemma 2 $\forall k, \|\mathbf{v}_k\|^2 \leq kR$. *In other words, the norm of \mathbf{v}_k grows “slowly”, at a rate depending on R .*

Proof:

$$\begin{aligned}
& \mathbf{v}_{k+1} \cdot \mathbf{v}_{k+1} = (\mathbf{v}_k + y_i \mathbf{x}_i) \cdot (\mathbf{v}_k + y_i \mathbf{x}_i) \\
\Rightarrow & \|\mathbf{v}_{k+1}\|^2 = \|\mathbf{v}_k\|^2 + 2y_i \mathbf{x}_i \cdot \mathbf{v}_k + y_i^2 \|\mathbf{x}_i\|^2 \\
\Rightarrow & \|\mathbf{v}_{k+1}\|^2 = \|\mathbf{v}_k\|^2 + [\text{something negative}] + 1\|\mathbf{x}_i\|^2 \\
\Rightarrow & \|\mathbf{v}_{k+1}\|^2 \leq \|\mathbf{v}_k\|^2 + \|\mathbf{x}_i\|^2 \\
\Rightarrow & \|\mathbf{v}_{k+1}\|^2 \leq \|\mathbf{v}_k\|^2 + R \\
\Rightarrow & \|\mathbf{v}_k\|^2 \leq kR
\end{aligned}$$

These steps follow from: “B’s strategy”; distributivity; “B’s strategy” again (in particular, the fact that there was a mistake, and the fact that $y_i \in \{-1, +1\}$); a simplification; the “radius R ” assumption; and induction.

Finally, we can wrap this up and prove the theorem. Notice that Lemma 2 bounds the norm of \mathbf{v}_k , while Lemma 1 bounds the dot product of \mathbf{v}_k and \mathbf{u} . Think about this a second: if the norm of \mathbf{v}_k is small and the dot product with \mathbf{u} is large, then \mathbf{v}_k might be converging to \mathbf{u} —but it depends on the details of how fast these different quantities grow. So the question is, how fast two these trends go? How do we compare them?

It’s not obvious, but the way to combine these is to start by squaring the inequality from Lemma 1:

$$\begin{aligned}
& (k\gamma)^2 \leq (\mathbf{v}_k \cdot \mathbf{u})^2 \\
\Rightarrow & k^2 \gamma^2 \leq \|\mathbf{v}_k\|^2 \|\mathbf{u}\|^2 \\
\Rightarrow & k^2 \gamma^2 \leq \|\mathbf{v}_k\|^2
\end{aligned}$$

The last step follows since conveniently, $\|\mathbf{u}\| = 1$. Now we have a bound on k^2 in terms of the norm of \mathbf{v}_k . Now put this together with Lemma 2, which has a bound on \mathbf{v}_k in terms of kR .

$$k^2\gamma^2 \leq \|\mathbf{v}_k\|^2 \leq kR$$

How sweet—now we have a quadratic function in k that’s upper-bounded by a linear function in k . That certainly ought to lead to a bound on k , right? To follow through...

$$\begin{aligned} \Rightarrow k^2\gamma^2 &\leq kR \\ \Rightarrow k\gamma^2 &\leq R \\ \Rightarrow k &\leq \frac{R}{\gamma^2} \end{aligned}$$

And that’s the whole proof for the mistake bound.

2 Converting to batch

Interestingly, we’ve shown that B won’t make too many mistakes, but we haven’t said anything about how accurate the “last” guess $v_{k_{max}}$ is on i.i.d data. When the data is separable, it seems like it should be ok—but the mistake-bound analysis can be extended to handle non-separable data as well, and in that case it’s not at all clear what we should do.

Imagine we run the on-line perceptron and see this result.

i	guess	input	result
1	\mathbf{v}_0	\mathbf{x}_1	X (a mistake)
2	\mathbf{v}_1	\mathbf{x}_2	✓ (correct!)
3	\mathbf{v}_1	\mathbf{x}_3	✓
4	\mathbf{v}_1	\mathbf{x}_4	X (a mistake)
5	\mathbf{v}_2	\mathbf{x}_5	✓
6	\mathbf{v}_2	\mathbf{x}_6	✓
7	\mathbf{v}_2	\mathbf{x}_7	✓
8	\mathbf{v}_2	\mathbf{x}_8	X
9	\mathbf{v}_3	\mathbf{x}_9	✓
10	\mathbf{v}_3	\mathbf{x}_{10}	X

Our final hypothesis could be the most accurate guess constructed so far, which looks to be \mathbf{v}_2 (which was right 3 times and before it had an error, for an empirical error rate of 25%). Or we could use the last guess \mathbf{v}_4 , which we haven’t tested at all...

Rob and Yoav suggest using a mixture of all of these, weighting them by their accuracy on the data—or more precisely, by m_k , the number of examples they were used for. In this case, we'd randomly pick \mathbf{v}_0 with probability $1/10$, \mathbf{v}_1 with probability $3/10$, \mathbf{v}_2 with probability $4/10$, and \mathbf{v}_3 with probability $2/10$. After picking our \mathbf{v}_k , we just use its prediction.

Each \mathbf{v}_k 's empirical error is $1/m_k$, so if we let $m = \sum m_k$ then the probability of an error using this voting scheme is

$$\begin{aligned} P(\text{error in } \mathbf{x}) &= \sum_k P(\text{error on } \mathbf{x} | \text{picked } \mathbf{v}_k) P(\text{picked } \mathbf{v}_k) \\ &= \sum_k \frac{1}{m_k} \frac{m_k}{m} = \sum_k \frac{1}{m} = \frac{k}{m} \end{aligned}$$

This is easy to understand and seems to work well experimentally.

Even easier is to approximate the voting with a weighted average, and classify with a single vector

$$\mathbf{v}_* = \sum_k \left(\frac{m_k}{m} \mathbf{v}_k \right)$$

3 Online learning for ranking

Here's a very similar on-line model for learning to *rank* items.

An *on-line learner* L again has two methods:

- $L.\text{findBest}(\mathbf{x}_1, \dots, \mathbf{x}_n)$ returns an integer $\hat{\ell}$ between 1 and n . Here $\hat{\ell}$ is L 's prediction as to which \mathbf{x}_j is “best”—i.e, a prediction that $\mathbf{x}_{\hat{\ell}}$ should be ranked highest.
- $L.\text{addExample}(\mathbf{x}_1, \dots, \mathbf{x}_n, \ell)$ is called to “tell” L that \mathbf{x}_ℓ is the true “best” element of the given list of \mathbf{x}_j 's.

Again, we can imagine a game. On each the i -th round, A generates a list $\mathbf{x}_{i,1}, \dots, \mathbf{x}_{i,n_i}$; B makes a prediction $\hat{\ell}$; and then A provides the index ℓ of the “true” best list element \mathbf{x}_ℓ . B makes a mistake iff $\hat{\ell} \neq \ell$.

3.1 The perceptron game

The rules for the “perceptron ranking game” are:

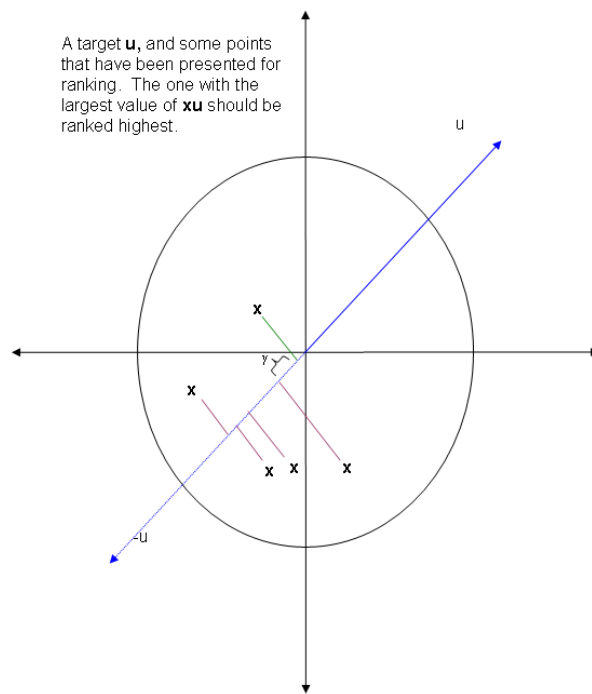


Figure 2: A perceptron used for ranking

Margin γ . A must provide examples that can be correctly ranked with some vector \mathbf{u} with margin $\gamma > 0$, ie

$$\exists \mathbf{u} : \forall \mathbf{x}_{i,1}, \dots, \mathbf{x}_{i,n_i}, \ell \text{ given by } A, \forall j \neq \ell, \mathbf{u} \cdot \mathbf{x}_\ell - \mathbf{u} \cdot \mathbf{x}_j > \gamma$$

and furthermore, $\|\mathbf{u}\|^2 = 1$.

Radius R . A must provide examples “near the origin”, ie

$$\forall \mathbf{x}_i \text{ given by } A, \|\mathbf{x}\|^2 < R$$

B 's strategy. B uses this learning strategy.

1. B 's initial guess is $\mathbf{v}_0 = \mathbf{0}$.
2. At every round, B 's prediction is the j such that $\mathbf{x}_{i,j}$ that has the highest “score” as computed by $score = \mathbf{v}_k \cdot \mathbf{x}_{i,j}$, where \mathbf{v}_k is the current guess. I.e.,

$$\hat{\ell} = \operatorname{argmax}_j \mathbf{v}_k \cdot \mathbf{x}_{i,j}$$

3. When B makes a mistake then B updates the guess as follows:

$$\mathbf{v}_{k+1} = \mathbf{v}_k + \mathbf{x}_{i,\ell} - \mathbf{x}_{i,\hat{\ell}}$$

3.2 The analysis

The analysis is almost exactly the same as before. If you recall, the key things about $y_i \mathbf{x}_i$ we needed to use were that

- for lemma 1, $y_i \mathbf{x}_i \cdot \mathbf{u} > \gamma$;
- for lemma 2, $y_i \mathbf{x}_i \cdot \mathbf{v}_k < 0$ when \mathbf{v}_k made a mistake on \mathbf{x}_i .

The ranking perceptron's analysis uses the same properties for $(\mathbf{x}_\ell - \mathbf{x}_{\hat{\ell}})$.

- for lemma 1, we'll use $(\mathbf{x}_\ell - \mathbf{x}_{\hat{\ell}}) \cdot \mathbf{u} > \gamma$;
- for lemma 2, we'll use $(\mathbf{x}_\ell - \mathbf{x}_{\hat{\ell}}) \cdot \mathbf{v}_k < 0$, when there's a mistake made on \mathbf{x} by \mathbf{v}_k .

Follow along:

Lemma 3 $\forall k, \mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$.

Proof:

$$\begin{aligned}
& \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k + \mathbf{x}_{i,\ell} - \mathbf{x}_{i,\hat{\ell}}) \cdot \mathbf{u} \\
\Rightarrow & \mathbf{v}_{k+1} \cdot \mathbf{u} = \mathbf{v}_k \cdot \mathbf{u} + \mathbf{x}_{i,\ell} \cdot \mathbf{u} - \mathbf{x}_{i,\hat{\ell}} \cdot \mathbf{u} \\
\Rightarrow & \mathbf{v}_{k+1} \cdot \mathbf{u} \geq \mathbf{v}_k \cdot \mathbf{u} + \gamma \\
\Rightarrow & \mathbf{v}_k \cdot \mathbf{u} \geq k\gamma
\end{aligned}$$

Here the margin assumption means that $\mathbf{x}_{i,\ell} \cdot \mathbf{u} - \mathbf{x}_{i,\hat{\ell}} \cdot \mathbf{u} > \gamma$

Lemma 4 $\forall k, \|\mathbf{v}_k\|^2 \leq 2kR$.

Proof (a little more involved than before):

$$\begin{aligned}
& \mathbf{v}_{k+1} \cdot \mathbf{v}_{k+1} = (\mathbf{v}_k + (\mathbf{x}_{i,\ell} - \mathbf{x}_{i,\hat{\ell}})) \cdot (\mathbf{v}_k + (\mathbf{x}_{i,\ell} - \mathbf{x}_{i,\hat{\ell}})) \\
\Rightarrow & \|\mathbf{v}_{k+1}\|^2 = \|\mathbf{v}_k\|^2 + 2(\mathbf{v}_k \cdot \mathbf{x}_{i,\ell} - \mathbf{v}_k \cdot \mathbf{x}_{i,\hat{\ell}}) + \|\mathbf{x}_{i,\ell} - \mathbf{x}_{i,\hat{\ell}}\|^2
\end{aligned}$$

So far, that's just algebra. Next use the fact that there was a mistake, and hence $\mathbf{v}_k \cdot \mathbf{x}_{i,\ell} - \mathbf{v}_k \cdot \mathbf{x}_{i,\hat{\ell}} < 0$, and then the ‘‘triangle inequality’’ which says that $\|\mathbf{x} + \mathbf{y}\|^2 \leq \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2$, and finally the radius assumption:

$$\begin{aligned}
& \|\mathbf{v}_{k+1}\|^2 = \|\mathbf{v}_k\|^2 + 2(\mathbf{v}_k \cdot \mathbf{x}_{i,\ell} - \mathbf{v}_k \cdot \mathbf{x}_{i,\hat{\ell}}) + \|\mathbf{x}_{i,\ell} - \mathbf{x}_{i,\hat{\ell}}\|^2 \\
\Rightarrow & \|\mathbf{v}_{k+1}\|^2 \leq \|\mathbf{v}_k\|^2 + \|\mathbf{x}_{i,\ell} - \mathbf{x}_{i,\hat{\ell}}\|^2 \\
\Rightarrow & \|\mathbf{v}_{k+1}\|^2 \leq \|\mathbf{v}_k\|^2 + \|\mathbf{x}_{i,\ell}\|^2 + \|\mathbf{x}_{i,\hat{\ell}}\|^2 \\
\Rightarrow & \|\mathbf{v}_{k+1}\|^2 \leq \|\mathbf{v}_k\|^2 + 2R
\end{aligned}$$

So $\|\mathbf{v}_k\|^2 \leq 2kR$ by induction. Putting these two lemmas together gives a bound, as in the classification case.

Theorem 2 *Under the rules of the ranking perceptron game, it is always the case that $k < 2R/\gamma^2$.*

4 Online learning for sequential classification

Now here a variant of the game, which gets us even closer to NER. On each round:

1. A constructs (or finds) a sequence $a_1 \dots a_n$ —say, the words from an n -word news article.
2. If each word can be labeled positive (for “part of an entity”) or negative, then associated with that sequence is a set of 2^n possible label sequences each sequence being a length- n vector containing the labels of every word in the article.

A next creates 2^n objects $\mathbf{x}_1, \dots, \mathbf{x}_{2^n}$, where \mathbf{x}_j is a *vector of features* extracted from the pair $((a_1 \dots a_n), (y_1^j \dots y_n^j))$, where $(y_1^j \dots y_n^j)$ is the j -th possible sequence of labels (from that enormous set of 2^n possible label sequences) for $a_1 \dots a_n$.

3. After creating these 2^n vectors $\mathbf{x}_1, \dots, \mathbf{x}_{2^n}$, A passes them to B for a prediction.
4. B returns a prediction $\hat{\ell}$, using the voted perceptron for ranking above. That is, B finds the $\hat{\ell}$ such that the score of $\mathbf{v}_k \cdot \mathbf{x}_{\hat{\ell}}$ is highest, for B 's current guess \mathbf{v}_k .
5. A tells B which \mathbf{x}_{ℓ} really should be ranked highest. Since each \mathbf{x}_j corresponds to some sequence of labels $y_1^j \dots, y_n^j$ for the words in the article, this is equivalent to A telling B the “best” labeling for the article.

Just as in the ranking perceptron game, there has to be some unit \mathbf{u} that makes A 's intended ranking consistent: i.e., $\mathbf{u} \cdot \mathbf{x}_{\ell} - \mathbf{u} \cdot \mathbf{x}_j > \gamma$. In this case the \mathbf{u} vector has give the highest score to the featurized version of the pair $((a_1 \dots a_n), (y_1^j \dots, y_n^j))$.

There's no reason in principle why this couldn't work—if the feature vectors are sparse then the \mathbf{x}_j 's will have small radius, and if the features are good then a consistent ranking should be possible.

However, A and B couldn't really play this game efficiently because the lists of \mathbf{x} 's are too long.

But, A doesn't need to send the list of \mathbf{x} 's to B — B could construct them on-the-fly when A simply sends him the article $a_1 \dots a_n$. In fact, B doesn't need to construct them either—all B needs to do is pick a single best $\hat{\ell}$, which corresponds to single labeling $y_1^{\hat{\ell}} \dots, y_n^{\hat{\ell}}$, and if the features are all Markovian, then B can just use Viterbi to find that labels gives the highest score.

To summarize, if $\Phi(a_1 \dots a_n, y_1 \dots y_n)$ produces the featurizes vector \mathbf{x} , then the game above is equivalent to this one.

1. Initially B lets $\mathbf{v}_0 = \mathbf{0}$, and lets $k = 0$.
2. On each round of the game:
 - (a) A finds a sequence $a_1 \dots a_n$ and sends it to B .
 - (b) B uses Viterbi to find the sequence of labels $\hat{y}_1 \dots \hat{y}_n$ such that $\mathbf{v}_k \cdot \Phi(a_1 \dots a_n, \hat{y}_1 \dots \hat{y}_n)$ is largest, i.e., larger than $\mathbf{v}_k \cdot \Phi(a_1 \dots a_n, y'_1 \dots y'_n)$ for any other label sequence $y'_1 \dots y'_n$.
 - (c) A tells B the best labels $y_1 \dots y_n$ for $a_1 \dots a_n$.
 - (d) If B 's guess was wrong then B lets

$$\mathbf{v}_{k+1} = \mathbf{v}_k + \Phi(a_1 \dots a_n, y_1 \dots y_n) - \Phi(a_1 \dots a_n, \hat{y}_1 \dots \hat{y}_n)$$

and increments k (so that \mathbf{v}_{k+1} is used for subsequent predictions).

Comment: if we replace $a_1 \dots a_n$ with \mathbf{x} , $y_1 \dots y_n$ with \mathbf{y} , and Φ with $F(\mathbf{x}, \mathbf{y})$, then we're back to our notation for CRFs.

5 The kernel trick

Notice that \mathbf{v}_k is always the sum of certain x_i 's (namely, the ones where a mistake was made). This means that we can rewrite the prediction in terms of these x_i 's. Let \mathcal{M}_k be the first k indices i where a mistake was made: then

$$\mathbf{v}_k = \sum_{i \in \mathcal{M}_k} y_i \mathbf{x}_i$$

so the prediction made on some test example \mathbf{x} would be

$$\mathbf{v}_k \cdot \mathbf{x} = \left(\sum_{i \in \mathcal{M}_k} y_i \mathbf{x}_i \right) \cdot \mathbf{x} = \sum_{i \in \mathcal{M}_k} y_i (\mathbf{x}_i \cdot \mathbf{x}) = \sum_{i \in \mathcal{M}_k} y_i K(\mathbf{x}_i, \mathbf{x})$$

where $K(\mathbf{x}_i, \mathbf{x})$ is defined, for now, as the dot product. In other words, we don't need to compute \mathbf{v}_k : we could just keep around the \mathbf{x}_i 's in \mathcal{M}_k , and their associated (+1/-1) labels y_i . Implemented this way, and thinking of $K(\cdot, \cdot)$ as a measure of similarity, the perceptron algorithm is a sort of weighted nearest neighbor algorithm.

This suggests we could plug in some other similarity function and get reasonable results. The “kernel trick” is a mathematical trick that shows one can in which we can: basically, whenever $K(\mathbf{x}_i, \mathbf{x})$ is equivalent to the dot product of some transformed vectors $H(\mathbf{x}_i)$ and $H(\mathbf{x})$.

As an example: suppose we transform $\mathbf{x} = \langle x_1, \dots, x_n \rangle$ by adding in all the n^2 “interaction terms”: $x_1x_1, x_1x_2, \dots, x_nx_n$, plus another component that will always be equal to “1”: i.e.,

$$H'(\langle x_1, \dots, x_n \rangle) = \langle x_1x_1, x_1x_2, \dots, x_nx_n, x_1 \dots x_n, 1 \rangle$$

Consider the function $K(\mathbf{v}, \mathbf{x}) = (\mathbf{v} \cdot \mathbf{x} + 1)^2$.

$$\begin{aligned} K(\mathbf{v}, \mathbf{x}) &= (\mathbf{v} \cdot \mathbf{x} + 1)(\mathbf{v} \cdot \mathbf{x} + 1) \\ &= (\mathbf{v}\mathbf{x})^2 + 2\mathbf{v}\mathbf{x} + 1 \\ &= (v_1x_1 + \dots + v_nx_n)^2 + 2(v_1x_1 + \dots + v_nx_n) + 1 \\ &= \sum_{i,j} v_i x_i v_j x_j + 2 \sum_i v_i x_i + 1 \\ &= \sum_{i,j} v_i v_j x_i x_j + 2 \sum_i v_i x_i + 1 \end{aligned}$$

This is almost the same as $H'(\mathbf{v}) \cdot H'(\mathbf{x})$ - the only difference is the extra factor of 2 in the linear terms, which we could fix by using H as

$$H(\langle x_1, \dots, x_n \rangle) = \langle x_1x_1, x_1x_2, \dots, x_nx_n, \sqrt{2}x_1 \dots \sqrt{2}x_n, 1 \rangle$$

Then $K(\mathbf{x}, \mathbf{y}) = H(\mathbf{x}) \cdot H(\mathbf{y})$.

So, using the “nearest-neighbor like” implementation of perceptron with the $K(\cdot, \cdot)$ function above is just like (equivalent to!) learning in transformed space of $H(\mathbf{x})$'s.