Statistical Techniques in Robotics (16-831, S20) Lecture #05 (Wednesday, February 02)

Online Linear Classification

Lecturer: Kris Kitani *Scribe: Daphne Chen*

1 Review

Previously in lecture we covered the Weighted Majority Algorithm (WMA) and Randomized Weighted Majority Algorithm (RWMA), and discussed the principle of using regret in an algorithm. We also learned about other forms of online learning algorithms, of which Prediction With Expert Advice (PWEA) and Online Linear Classification (OLC) are both a subset. In this section, we will briefly summarize the concepts from the prior lectures to help set the stage for discussing OLC in greater depth.

1.1 Weighted Majority Algorithm

WMA works by learning a weight for each of the expert algorithms, then aggregates them together to determine the prediction with the highest score. Over time, the expert that tends to make the best predictions (and thus has the highest weight) will have its weight proportionately increase, and will accordingly become the trusted opinion amongst the algorithms.

The full algorithm consists of the following steps:

Algorithm 1 Weighted Majority Algorithm

1: **function** WEIGHTED MAJORITY ALGORITHM
2: $\boldsymbol{w}^{(1)} \leftarrow \{w_n^{(1)} = 1\}_{n=1}^{N}$ ▷ Initialize weights
3: $\eta \leq \frac{1}{2}$ ▷ Initialize penalty rate parameter
4: **for** $t = 1, \cdots, T$ **do**
5: RECEIVE $(\boldsymbol{x}^{(t)} \in \{-1, 1\}^N)$ ▷ Receive expert predictions as observations from -1 to 1
6: $\hat{y}^{(t)} = \text{sign}\Big(\sum_{n=1}^{N} x_n^{(t)} \cdot w_n^{(t)}\Big) \in \{-1, 1\}$ ▷ Make prediction as binary output
7: RECEIVE $(y^{(t)} \in \{-1, 1\})$ ▷ Get actual prediction
8: $w_n^{(t+1)} \leftarrow w_n^{(t)}\big(1 - \eta \cdot \mathbf{1}[y^{(t)} \neq x_n^{(t)}]\big)$ ▷ Weight penalty update
9: **end for**

It is also key to note the update equation (line 8 of the algorithm), which shows that the weights are updated when the learner prediction and the actual prediction do not match. This can be used to derive the regret bound.

Accordingly, we utilize cumulative loss through regret, given by $R^{(T)}(H)$. This parameter is bounded by

$$R(h_n) = M^{(T)} - m_n^{(T)} \leq (1 + 2\eta)m_n^{(T)} + \frac{2\log N}{\eta} \tag{1}$$

By studying this upper bound for regret, we can conclude that the average regret does not converge to zero. Thus WMA is not a no-regret algorithm. Instead, the algorithm has bounded regret, where error scales linearly with time.

1.2 Randomized Weighted Majority Algorithm

In contrast to WMA, as the name suggests, the RWMA approach works by adding randomization by sampling a single expert rather than weighting together multiple expert opinions. Similarly, it is also a no-regret algorithm. We will see in the following outline of the algorithm how adding one step can cut the number of mistakes in half.

The full algorithm for RWMA consists of the following steps:

Algorithm 2 Randomized Weighted Majority Algorithm (RWMA)

function RANDOMIZED WEIGHTED MAJORITY ALGORITHM
$\mathbf{w}^{(1)} \leftarrow \{w_n^{(1)} = 1\}_{n=1}^{N}$ ▷ Initialize weights
$\eta \leq \frac{1}{2}$ ▷ Initialize penalty rate parameter
for $t = 1, \cdots, T$ **do**
  RECEIVE $(\mathbf{x}^{(t)} \in \{-1, 1\}^N)$ ▷ Receive expert predictions as observations from -1 to 1
  $I \sim \text{MULTINOMIAL}(\mathbf{w}^{(t)}/\Phi^{(t)})$, where $\Phi^{(t)} = \sum_{n=1}^{N} w_n^{(t)}$
  $\hat{y}^{(t)} = h_i(\mathbf{x}^{(t)})$ ▷ Make prediction as binary output
  RECEIVE $(y^{(t)} \in \{-1, 1\})$ ▷ Get actual prediction
  $w_n^{(t+1)} \leftarrow w_n^{(t)}\big(1 - \eta \cdot \mathbf{1}[y^{(t)} \neq h_n(\mathbf{x})^{(t)}]\big)$ ▷ Weight penalty update
end for

Here, the expected regret is bounded by:

$$\mathbb{E}[R] \leq \eta m_n^{(T)} + \frac{\log N}{\eta} \tag{2}$$

2 Summary

2.1 Online Linear Classification

We will begin by summarizing Online Linear Classification (OLC) before moving on to the following sections which dive into the two algorithms, Perceptron and Winnow, that are used to learn an online linear classifier.

OLC is a type of Online Learning algorithm, similar to PWEA, as we covered in prior lectures. Let us first briefly recap linear classification to set the stage for discussing OLC. Most linear classifiers use all data at once during the training process, whereas online learning requires that the training samples are processed one by one as data becomes available.

Most linear classification problems can be set up as a decision problem based on the linear combination of input features. In the example used in this class, this took form as a method of fitting a

hyperplane to classify positive and negative examples into their correct classes by determining the appropriate prediction rule.

From here, we will discuss two approaches for learning an online linear classifier:

1. The Perceptron algorithm, which uses additive updates, and
2. The Winnow algorithm, which uses multiplicative updates.

3 Perceptron Algorithm

3.1 Summary

The Perceptron Algorithm is the oldest and one of the most classic approaches for minimizing regret in OLC. In summary, it works by updating the current prediction $x^{(t)}$ as denoted in Line 7 in Algorithm 3 below once a mistake is detected.

The steps for the Perceptron Algorithm are shown below:

Algorithm 3 Perceptron Algorithm

1: **function** PERCEPTRON ALGORITHM
2: $\mathbf{w}^{(1)} \leftarrow 0$ ▷ Initialize weights
3: **for** $t = 1, \cdots, T$ **do**
4: RECEIVE $(\mathbf{x}^{(t)} \in R^N)$ ▷ Receive expert predictions
5: $\hat{y}^{(t)} = \text{sign}\Big(w^{(t)} x^{(t)}\Big)$ ▷ Make prediction
6: RECEIVE $(y^{(t)} \in \{-1, 1\})$ ▷ Get actual prediction
7: $w_n^{(t+1)} \leftarrow w_n^{(t)} + y^{(t)} \cdot x^{(t)} \cdot \mathbf{1}[y^{(t)} \neq x_n^{(t)}]$ ▷ Weight penalty update
8: **end for**

As noted in the lecture, the Perceptron Algorithm possesses the following properties:

1. It is fast, because the update for an incorrect prediction is a dot product and the update is a sum, thus can run in linear time. No weight updates are required for correct predictions, further simplifying the algorithm.
2. It is not a large margin classifier, since it only uses sign for making predictions, thus the algorithm contains no notion of margin.
3. The algorithm does not work on non-separable data; since the Perceptron Algorithm defines its decision boundary with a *linear* hyperplane, it is only able to classify linearly separable data.

3.2 Mistake Bound

In this section, we will derive the mistake bound for the Perceptron Algorithm using the following 5-step strategy:

1. Define a potential function
2. Upper bound the potential function
3. Lower bound the potential function
4. Combine bounds
5. Get performance bound using algebra or approximation

Theorem 1. *(Perceptron Mistake Bound) Let R be the norm of observations, where $R = \max_t \|x^{(t)}\|$. Let γ be the margin of separability, where $\gamma = \min_t y_t \langle \boldsymbol{w}^\star, \boldsymbol{x}^{(t)} \rangle$ and $\|\boldsymbol{w}^\star\| = 1$. Then the Perceptron Mistake Bound is as follows where M is the total mistakes made by the Perceptron Algorithm:*

$$M \leq \frac{R^2}{\gamma^2} \tag{3}$$

Proof. Following the 5 steps defined above, we will begin by defining the potential function, in this case as the squared l_2 norm of the weight vector $\mathbf{w}$:

$$\Phi^{(t)} = ||\boldsymbol{w}^{(t)}||^2 = \sum_{n=1} \left(w_n^{(t)}\right)^2 \tag{4}$$

Next, we will use algebra to manipulate the equation into a form where we will derive the upper bound.

$$\begin{aligned}
\Phi^{(t)} &= ||\boldsymbol{w}^{(t-1)} + y^t \boldsymbol{x}^{(t)}||^2 = \sum_{n=1} \left(w_n^{(t)}\right)^2, && (5)\\
&= ||\boldsymbol{w}^{(t-1)}||^2 + ||\boldsymbol{x}||^2 + 2y^{(t)} \langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^t \rangle && (6)\\
&\leq ||\boldsymbol{w}^{(t-1)}||^2 + ||\boldsymbol{x}||^2, && (7)\\
&\leq ||\boldsymbol{w}^{(t-1)}||^2 + R^2 \quad (\text{Where } R = \max_t \|x^{(t)}\|). && (8)
\end{aligned}$$

Next, we will determine the upper bound using induction as follows, beginning from the base case:

$$\begin{aligned}
\Phi^{(1)} &\leq ||\boldsymbol{w}^{(0)}||^2 + R^2, \qquad (1 \text{ mistakes}) && (9)\\
\Phi^{(2)} &\leq ||\boldsymbol{w}^{(0)}||^2 + 2R^2, \qquad (2 \text{ mistakes}) && (10)\\
&\vdots && (11)\\
\Phi^{(T)} &\leq ||\boldsymbol{w}^{(0)}||^2 + M^{(T)} R^2, \quad (\text{M mistakes}) && (12)\\
\Phi^{(T)} &\leq M^{(T)} R^2 && (13)
\end{aligned}$$

Now that we have upper bounded the potential function, we can proceed with the next step, lower bounding the potential function. We will do this by starting with an intermediate potential function given by the following:

$$\left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(t)} \right\rangle \tag{14}$$

In Equation 14, the symbol $\boldsymbol{w}^*$ is used to indicate a perfect classifier which is a unit vector (i.e. its norm is equal to 1), while the symbol $\boldsymbol{w}^{(t)}$ represents the classifier at timestep t.

First, let us derive an upper bound for Equation 14:

$$\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} = \arg\max_{\boldsymbol{w}'} \left\langle \boldsymbol{w}', \boldsymbol{w} \right\rangle \tag{15}$$

We start from Equation 15, following the logic that the dot product in Equation 14 will be maximized when the values $\boldsymbol{w}^*$ and $\boldsymbol{w}$ are in the same orientation.

The upper bound is given by:

$$\left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(T)} \right\rangle \leq \left\langle \frac{\boldsymbol{w}^{(T)}}{\left\|\boldsymbol{w}^{(T)}\right\|}, \boldsymbol{w}^{(T)} \right\rangle = \frac{\left\|\boldsymbol{w}^{(T)}\right\|^2}{\left\|\boldsymbol{w}^{(T)}\right\|} = \left\|\boldsymbol{w}^{(T)}\right\| \tag{16}$$

Next, we will derive a lower bound for Equation 14:

After applying the update rule, we get:

$$\left\langle \boldsymbol{w}^* \boldsymbol{w}^{(t)} \right\rangle = \left\langle \boldsymbol{w}^* \boldsymbol{w}^{(t-1)} \right\rangle + y^{(t)} \left\langle \boldsymbol{w}^* \boldsymbol{w}^{(t)} \right\rangle , \tag{17}$$

$$\left\langle \boldsymbol{w}^* \boldsymbol{w}^{(t)} \right\rangle \geq \left\langle \boldsymbol{w}^* \boldsymbol{w}^{(t-1)} \right\rangle + \gamma , \tag{18}$$

After plugging in $\gamma = \min_x y_t \boldsymbol{w}^* \boldsymbol{x}^{(t)}$.

Once again using induction to derive the lower bound, starting from the base case:

$$\begin{aligned}
\left\langle \boldsymbol{w}^* \boldsymbol{w}^{(1)} \right\rangle &\geq \left\langle \boldsymbol{w}^* \boldsymbol{w}^{(0)} \right\rangle + \gamma , \\
\left\langle \boldsymbol{w}^* \boldsymbol{w}^{(2)} \right\rangle &\geq \left\langle \boldsymbol{w}^* \boldsymbol{w}^{(0)} \right\rangle + 2\gamma , \\
&\vdots \\
\left\langle \boldsymbol{w}^* \boldsymbol{w}^{(T)} \right\rangle &\geq \left\langle \boldsymbol{w}^* \boldsymbol{w}^{(0)} \right\rangle + M^{(T)}\gamma .
\end{aligned} \tag{19}$$

Thus we have derived that the lower bound for Equation 14 is

$$\left\langle \boldsymbol{w}^* \boldsymbol{w}^{(T)} \right\rangle \geq M^{(T)} \gamma \,. \tag{20}$$

We can then move on to step 4 of the 5-step strategy, combining bounds. We are able to do that as follows using the results derived in steps 2 and 3 above, in Equations 16 and 20, respectively. In doing so, we will derive the lower bound of the potential function. Thus:

$$M^T \cdot \gamma \leq \left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(T)} \right\rangle \leq \left\| \boldsymbol{w}^{(T)} \right\| \tag{21}$$

$$M^T \cdot \gamma \leq \left\| \boldsymbol{w}^{(T)} \right\| \tag{22}$$

$$\left(M^T \cdot \gamma\right)^2 \leq \left\| \boldsymbol{w}^{(T)} \right\|^2 \text{ squaring both sides, where } \left\| \boldsymbol{w}^{(T)} \right\|^2 \text{is the potential function we defined} \tag{23}$$

The last step of the 5-step approach is to get the mistake bound of the Perceptron Algorithm. Since we have derived the upper and lower bounds of the potential function, we can combine them as follows to get the mistake bound:

$$\begin{aligned} \left(M^T \cdot \gamma\right)^2 &\leq M^{(T)} \cdot R^2 \\ \therefore M^{(T)} &\leq \frac{R^2}{\gamma^2} \end{aligned} \tag{24}$$

From this mistake bound we can deduce that when γ is large, M becomes smaller since the data is more separable, thus the learner is better able to classify the data.

4 Winnow Algorithm

4.1 Summary

The second approach to OLC that we will study is the Winnow Algorithm. The Winnow Algorithm is different from the Perceptron Algorithm in that it uses a multiplicative weight update instead of additive. It represents the problem as a disjunctive boolean function, which supposes that not all input features are relevant; thus, it works on problems where both inputs and outputs are binary features/labels.

Further, it is different from the Perceptron Algorithm in that the Winnow Algorithm initializes the weights to 1 instead of 0 and makes predictions as binary output. Additionally, as noted above, the update step is exponential and multiplicative, as we can see below in Algorithm 4.

The steps for the Winnow Algorithm are shown below:

Algorithm 4 Winnow Algorithm

function WINNOW ALGORITHM
$\boldsymbol{w}^{(1)} = \{1, 1, ..., 1\}$ ▷ Initialize weights
for $t = 1, \cdots, T$ **do**
  RECEIVE $(\boldsymbol{x}^{(t)} \in \{0,1\}^N)$ ▷ Receive expert predictions
  $\hat{y}^{(t)} = \mathbf{1}[\langle w^{(t)}, x^{(t)} \rangle > N]$ ▷ Make prediction as binary output
  RECEIVE $(y^{(t)} \in \{0,1\})$
  $w_i^{(t+1)} = w_i^{(t)}(1+\beta)^{(y^{(t)} - \hat{y}^{(t)}) \cdot x_i^{(t)}}$ ▷ Weight update step
end for

Upon further analysis of the algorithm, we can note the following observations about the Winnow update rule – if there is an incorrect prediction, the value used to weight the attributes differs depending on the prediction. For instance, if the prediction is 0 and the true label is 1, then the multiplied value is $(1+\beta)$ (and thus the weight is increased); if vice versa, and the prediction is 1 and the true label is 0, then the multiplied value is $\frac{1}{(1+\beta)}$ (and thus the weight is decreased).

4.2 Mistake Bound

To dive further into our understanding of the Winnow Algorithm, we will perform an analysis of the mistake bound. As before, the first step is to define a potential function. We will define the following potential function as the sum of weights among all features at timestep t:

$$\Phi^{(t)} = \sum_{n=1}^{N} w_n^{(t)} \tag{25}$$

For every mistake m, we will separate out mistakes made on positive samples versus negative samples as follows:

$$m = m^+ + m^- \tag{26}$$

This allows us to appropriately bound the potential function. Starting from mistakes on positive samples,

$$\Phi^t = \Phi^{t-1} + \boldsymbol{w}^{(t-1)} \cdot \boldsymbol{x}^{(t-1)} \tag{27}$$

Where the weight is increased, as we noted above in our analysis. A mistake on a positive example means

$$\hat{y}^{(t)} = \mathbf{1}[\langle \mathbf{w}^{(t)}, \mathbf{x}^{(t)} \rangle > N] = 0 \tag{28}$$

$$\mathbf{w}^{(t-1)} \cdot \mathbf{x}^{(t-1)} \leq N \tag{29}$$

$$\Phi^{(t)} \leq \Phi^{(t-1)} + N \tag{30}$$

$$\Phi^{(t)} \leq \Phi^{1} + m^{+}N = N + m^{+}N \tag{31}$$

Where Equation 31, resulting from proof by induction, represents the upper bound on mistakes from positive samples.

Next, considering negative samples, weights are decreased by a factor of 2, thus:

$$\begin{aligned} \Phi^{(t)} &= \Phi^{(t-1)} - \sum_{n:x_n^{(t)}=1} \frac{1}{2} w_n^{(t-1)} \\ &= \Phi^{(t-1)} - \frac{1}{2}\mathbf{w}^{(t-1)} \cdot \mathbf{x}^{(t-1)} \end{aligned} \tag{32}$$

$$\Phi^{(t)} < \Phi^{(t-1)} - \frac{1}{2}N \tag{33}$$

$$\Phi^{(t)} < \Phi^{1} - \frac{1}{2}m^{-}N = N - m^{-}\frac{1}{2}N \tag{34}$$

Where Equation 34, resulting from proof by induction, represents the upper bound on mistakes from negative samples.

Finally, combining Equations 31 and 34, we can get the upper bound for the potential function:

$$\Phi^{(t)} \leq N + m^{+}N - m^{-}\frac{1}{2}N \tag{35}$$

Finding the lower bound is much more simple. Since the Winnow Algorithm uses weights initialized to 1 and the sign does not change during the algorithm update, the lower bound is simply 0.

Now, we combine the two:

$$0 \leq N + m^{+}N - m^{-}\frac{1}{2}N \tag{36}$$

$$m^{-} < 2 + 2m^{+} \tag{37}$$

To conclude our proof for the upper bound of the Winnow Algorithm, let us consider two lemmas: first, that if we make a mistake on a positive sample, then the weight $w_n^{(t)} < N$. Secondly, if a certain feature makes m mistakes, then its weight will be 2^m. This gives us the following:

$$w_n^{(t)} = 2^{m^{+}-1} < N \tag{38}$$

$$m^{+} < log_2 N + 1 \tag{39}$$

$$m^{+} < k(log_2 N + 1) \text{ (k relevant features)} \tag{40}$$

Now, use Equation 37 to get the upper bound of mistakes for the Winnow Algorithm:

$$m^- < 2 + 2k(log_2 N + 1) \tag{41}$$

$$m = m^+ + m^- < 2 + 3k(log_2 N + 1) \tag{42}$$

5 Conclusion

In this lecture we covered two types of Online Linear Classification: the Perceptron Algorithm and the Winnow Algorithm. While both approaches involve the notion of separability of data using a linear hyperplane, they are intrinsically different. The Perceptron Algorithm is fast because it has an additive weight update, while the Winnow Algorithm has an exponential, multiplicative update and thus performs better when there are irrelevant dimensions. We analyzed the steps of each algorithm and derived their mistake bounds. The content from this lecture will help set the stage for learning further about other forms of online learning.

References

[1] N. Littlestone, M. K. Warmuth. *The Weighted Majority Algorithm.*

[2] S. Arora. *Decision-making under total uncertainty: the multiplicative weight algorithm.*

[3] N. Littlestone, *Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow*, in Proceedings of the 4th Annual Conference on Computational Learning Theory, ACM, New York, 1991, pp. 147–156.