CS 4650/7650, Lecture 3
Discriminative Classification

Jacob Eisenstein

August 27, 2013

1 Recap

- Notation problems in reading
- Bag-of-words representation $\boldsymbol{f}(\boldsymbol{x}, y)$
- Classification as a dot-product $\boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}, y)$
- Naive Bayes
  - Define $P(\boldsymbol{x}, \boldsymbol{y})$ via a *generative model*
  - Prediction: $\hat{y} = \arg\max_y P(\boldsymbol{x}_i, y)$
  - Learning:

    $$\boldsymbol{w} = \arg\max_{\boldsymbol{w}} P(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \prod_i P(\boldsymbol{x}_i, y_i; \boldsymbol{w})\theta_{y,i} = \frac{\text{count}(Y = y, i)}{\text{count}(Y = y)}$$

    This gives the maximum-likelihood estimator (MLE; same as relative frequency estimator)
- Bias-variance tradeoff: MLE is high-variance, so add smoothing pseudo counts α. This reduces variance but adds bias.

2 A little more on NB

Another version of the Naive Bayes generative story:

- For each document i
  - Draw the label $y_i \sim \text{Categorical}(\theta)$
  - For each word $n \leq D_i$
    - Draw the word $\omega_{i,n} \sim \text{Categorical}(\phi_{y_i})$

This is really the same model as multinomial Naive Bayes, but it's a product of categorical distributions over words, instead of a multinomial distribution over word counts. The final probabilities are reduced by a factor corresponding to the normalization term in the multinomial, $\frac{(\sum_j x_j)!}{\prod_j x_j!}$.

This means we would generate the words in order, like $P(\textit{multinomial})P(\textit{Naive})P(\textit{Bayes})$.

2.1 The Naivety of Naive Bayes

Naive Bayes is very simple to work with. Estimation and prediction can done in closed form, and the nice probabilistic interpretation makes it relatively easy to extend the model in various ways.

But Naive Bayes makes assumptions which seriously limit its accuracy, especially in NLP.

- The multinomial distribution assumes that each word is generated independently of all the others (conditioned on the parameter ϕ_y). Formally, we assume conditional independence:

$$P(\textit{naïve}, \textit{Bayes}; \phi) = P(\textit{naïve}; \phi)P(\textit{Bayes}; \phi). \tag{1}$$

- But this is clearly wrong, because words "travel together." Question for you, is it:

$$P(\textit{naïve}, \textit{Bayes}) > P(\textit{naïve})P(\textit{Bayes}) \tag{2}$$

  or...

$$P(\textit{naïve}, \textit{Bayes}) < P(\textit{naïve})P(\textit{Bayes}) \tag{3}$$

  Apply the chain rule!

Traffic lights Dan Klein makes this point with an example about traffic lights. In his hometown of Pittsburgh, there is a 1/7 chance that the lights will be broken, and both lights will be red. There is a 3/7 chance that the lights will work, and the north-south lights will be green; there is a 3/7 chance that the lights work and the east-west lights are green.

The *prior* probability that the lights are broken is 1/7. If they are broken, the conditional likelihood of each light being red is 1. The prior for them not being broken is 6/7. If they are not broken, the conditional likelihood of each being light being red is 1/2.

Now, suppose you see that both lights are red. According to Naive Bayes, the probability that the lights are broken is $1/7 \times 1 \times 1 = 1/7 = 4/28$. The probability that the lights are not broken is $6/7 \times 1/2 \times 1/2 = 6/28$. So according to naive Bayes, there is a 60% chance that the lights are not broken!

What went wrong? We have made an independence assumption to factor the probability $P(R, R|\text{not-broken}) = P_{\text{north-south}}(R|\text{not-broken})P_{\text{east-west}}(R|\text{not-broken})$. But this independence assumption is clearly incorrect, because $P(R, R|\text{not-broken}) = 0$.

Less Naive Bayes? Of course we could decide not to make the naive Bayes assumption, and model $P(R, R)$ explicitly. But this idea does not scale when the feature space is large (as it often is in NLP). The number of possible feature configurations grows exponentially, so our ability to estimate accurate parameters will suffer from high variance. With an infinite amount of data, we'd be fine (in theory, maybe not in practice); but we never have that. Naive Bayes accepts some bias (because of the incorrect modeling assumption) in exchange for lower variance.

Features

- We have assumed "bag-of-words" features, which have relatively mild violations of the independence assumption.
- We may be interested in other features, which violate independence more severely. Can you think of any?
- Rich, overlapping features generally cannot be combined with Naive Bayes because the distortions resulting from violations of the independence assumption overwhelm the additional power of better features.
- To use these features, we will need other learning algorithms.

3 Perceptron

In NB, the weights can be interpreted as parameters of a probabilistic model. But this model requires an independence assumption that usually does not hold, and limits our choice of features.

Why not forget about probability and learn the weights in an error-driven way?

- Until converged, at each iteration t
  - Select an instance i
  - Let $\hat{y} = \arg\max_y \boldsymbol{w}_t^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y)$
  - If $\hat{y} = y_i$, do nothing
  - If $\hat{y} \neq y_i$, set $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \boldsymbol{f}(\boldsymbol{x}_i, \hat{y})$

Basically we are saying: if you make a mistake, increase the weights for features which are active with the correct label y_i, and decrease the weights for features which are active with the guessed label $\hat{y}$.

Theory:

- If there is a set of weights that correctly separates your data, then your data is **separable**.

- Formally, your data is separable if there exists a set of weights $\boldsymbol{w}$ such that

$$\forall \boldsymbol{x}_i, y_i,\ \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y_i) > \max_{y' \neq y_i} \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y') \tag{4}$$

- If your data is separable, it can be proven that the perceptron algorithm will eventually find a separator.
- What if your data is not separable?
  - the number of errors is bounded...
  - but the algorithm will thrash

3.1 Voted (averaged) perceptron

One solution to thrashing is to average the weights across all iterations:

$$\begin{aligned} \overline{\boldsymbol{w}} =& \frac{1}{T} \sum_t^T \boldsymbol{w}_t \\ y =& \arg\max_y \overline{\boldsymbol{w}}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}, y) \end{aligned}$$

There is some analysis showing that voting can improve generalization [4, 1]. However, this rule as described here is not practical. Can you see why? Can you see how to make it work?

4 Loss functions and large-margin classification

The perceptron is minimizing a **loss function**:

$$\ell_{\text{perceptron}}(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = \begin{cases} 0, & y_i = \arg\max_y \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(x_i, y) \\ 1, & \text{otherwise} \end{cases} \tag{5}$$

This isn't a great loss function. It's discontinuous and non-convex. Finding the global optimum is intractable when the data is not separable. We can define a loss function which behaves more nicely.

Let's define the *margin* as

$$\gamma(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(x_i, y_i) - \max_{y \neq y_i} \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y) \tag{6}$$

Then we can write a continuous "hinge loss" as

$$\ell_{\text{hinge}}(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = \begin{cases} 0, & \gamma(\boldsymbol{w}; \boldsymbol{x}_i, y_i) \geq 1, \\ 1 - \gamma(\boldsymbol{w}; \boldsymbol{x}_i, y_i), & \text{otherwise} \end{cases} \tag{7}$$

Equivalently, we can write $\ell_{\text{hinge}}(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = (1 - \gamma(\boldsymbol{w}; \boldsymbol{x}_i, y_i))_+$, where $(x)_+$ indicates the positive part of x.

Essentially, we want a *margin* of at least 1 between the score for the true label and the best-scoring alternative, which we have written $\hat{y}$.

4.1 Large-margin online classification: MIRA

Note that we can write $\boldsymbol{w} = s\boldsymbol{u}$, where $||\boldsymbol{u}||_2 = 1$. Think of s as the magnitude and $\boldsymbol{u}$ as the direction of the vector $\boldsymbol{w}$. If the data is separable, there are many values of s which attain zero hinge loss. For generality, we will try to make the smallest magnitude change to $\boldsymbol{w}$ possible.[1]

At step t, we optimize:

$$\arg\min_{w} \frac{1}{2}||\boldsymbol{w} - \boldsymbol{w}_t||^2 \ s.t. \ \ell_{\text{hinge}}(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = 0 \tag{8}$$

The solution to equation 8 is

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \tau_t(\boldsymbol{f}(y_i, \boldsymbol{x}_i) - \boldsymbol{f}(\hat{y}, \boldsymbol{x}_i)) \tag{9}$$

$$\tau_t = \frac{\ell(\boldsymbol{w}; \boldsymbol{x}_i, y_i)}{||\boldsymbol{f}(x_i, y_i) - \boldsymbol{f}(x_i, \hat{y})||^2}, \tag{10}$$

where again $\hat{y}$ is the best scoring y according to $\boldsymbol{w}_t$.

If the data is not separable, we can't meet the constraint in equation 8. To deal with this, we'll introduce a "slack" variable ξ_i. We use the slack variable to trade off between the constraint (having a large margin) and the objective (having a small change in $\boldsymbol{w}$). The tradeoff is controlled by a parameter C.

$$\arg\min_{w} \frac{1}{2}||\boldsymbol{w} - \boldsymbol{w}_t||^2 + C\xi_t \tag{11}$$

$$s.t. \ell_{\text{hinge}}(\boldsymbol{w}; \boldsymbol{x}_i, y_i) \leq \xi_t, \xi_t \geq 0 \tag{12}$$

The solution to 12 is,

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \tau_t(\boldsymbol{f}(y_i, \boldsymbol{x}_i) - \boldsymbol{f}(\hat{y}, \boldsymbol{x}_i)) \tag{13}$$

$$\tau_t = \min\left(C, \frac{\ell(\boldsymbol{w}; \boldsymbol{x}_i, y_i)}{||\boldsymbol{f}(x_i, y_i) - \boldsymbol{f}(x_i, \hat{y})||^2}\right), \tag{14}$$

- If C is 0, then infinite slack is permitted, and the weights will never change.
- As $C \to \infty$, no slack is permitted, and the optimization is identical to equation 8 and 10.

This algorithm is called MIRA, for Margin-Infused Relaxed Algorithm [2].

- MIRA has similar advantages to the averaged perceptron (stability, generalization).
- But MIRA allows more explicit control of the bias-variance tradeoff (large C = high variance, low bias).
- You can also apply weight averaging to MIRA.

[1] In the support vector machine (without slack variables), we choose the smallest magnitude weights that satisfy the constraint of zero hinge loss. Pegasos is an online algorithm for training SVMs [5]; it is similar to Passive-Aggressive.

4.2 Pros and cons of Perceptron and MIRA

- Perceptron and MIRA are error-driven, which means they usually do better in practice than naive Bayes.
- They are also online. This is a huge win: it means we can learn without having our whole dataset in memory at once.
- The original perceptron doesn't behave well if the data is not separable, and doesn't make it easy to control model complexity.
- All these models lack a probabilistic interpretation, which can be problematic if we want to add latent variables, perform joint learning, or handle missing data.

References

[1] M. Collins. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[2] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585, December 2006.

[3] K. Crammer and Y. Singer. Pranking with ranking. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *NIPS*, pages 641–647. MIT Press, 2001.

[4] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

[5] S. S. Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-GrAdient SOlver for SVM. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 807–814, New York, NY, USA, 2007. ACM.

[6] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

[7] A. Yessenalina and C. Cardie. Compositional matrix-space models for sentiment analysis. In *EMNLP*, pages 172–182. ACL, 2011.