Reading guide for Eistenstein (2019) ch. 2

LING83600

In ch. 2 Eistenstein describes—multinomial, unstructured, linear (non-neural)—classification algorithms commonly used in NLP, framing it as *linear text classification*, i.e., using bag-of-words features to classify a document.

2.1 Bag of Words

Eistenstein assumes that each text is represented by using a column vector of term counts (though one could easily use TF-IDF instead). He writes an example vector $x = [0, 1, 1, 0, 0, 2, 0, 1, 13, 0, \ldots]^\top$; here the bold indicates that x is a vector rather than a scalar, and $^\top$ is used to indicate that each document is represented as a column rather than a row vector.

In *multinomial text classification* we predict a label $\hat{y} \in \mathcal{Y}$ given a bag of words x using weights $\theta \in \mathbb{R}^{VK}$ where $K = |\mathcal{Y}|$ and V is the size of the vocabulary. We do this using a scoring function $\Psi(x, y)$ for each $y \in \mathcal{Y}$. The value of this scoring function, a real number, is a measure of the compatibility of the bag of words x with the label y. In a *linear classifier*, Ψ is defined by the inner product (i.e., the sum of products) of weights and a feature function $f(x, y)$

$$\Psi(x, y) = \theta \cdot f(x, y) = \sum_j \theta_j f_j(x, y).$$

Then, to predict, we use

$$\operatorname*{arg\,max}_{y \in \mathcal{Y}} \Psi(x, y);$$

that is, we select the label $\hat{y} \in \mathcal{Y}$ that gives the highest score. One could precompute θ using some prior information, but normally we use labeled data and optimization methods.

2.2 Naïve Bayes

Let the training set be $\{x^{(i)}, y^{(i)}\}_{i=1}^{N}$ where N is the number of examples. In *naïve Bayes* model, we *naïvely* assume that each example is independent. Then, the joint probability of the training set is given by

$$p(x, y) = \prod_{i=1}^{N} p(x^{(i)}, y^{(i)}).$$

To use the model for prediction, we have to define p and provide a *decision rule* for making a prediction using p. We assume that p is defined by two sets of parameters $\theta = \{\mu, \varphi\}$ such that $\mu \in [0,1]^K$ and $\varphi \in [0,1]^{K \times V}$. The estimates for these are given below.

2.2.1 Types and Tokens

Thus subsection describes a simple modification which *naïvely* assumes that each (word) count in x is independent of the others.

2.2.2 Prediction

It is relatively straightforward to use the joint probability distribution for classification. First, we can set θ using *maximum likelihood estimation* simply by counting the co-occurrences of x and y. Then, we can choose a label that maximizes the joint probability

$$\begin{aligned}\hat{y} &= \arg\max_{y \in \mathcal{Y}} p(x, y; \theta) \\ &= \arg\max_{y \in \mathcal{Y}} \left(\log p(x, y; \varphi) + \log p(y; \mu)\right)\end{aligned}$$

(By performing the computations in log-space we avoid the risk of numerical underflow.)

2.2.3 Estimation

We use *maximum likelihood estimation* to compute the parameters of this model, as follows:

$$\begin{aligned}\varphi_{y,j} &= \frac{c_{y,j}}{\sum_{j'=1}^{V} c_{y,j'}} \\ \mu_y &= \frac{c_y}{N}.\end{aligned}$$

2.2.4 Smoothing

> With text data, there is likely to be pairs of labels and words that never appear in the training set... (Eistenstein 2019:22)

This is a problem because the naïve Bayes formulation will assign zero probability to any labels. To avoid this, we often augment the counts $\varphi_{y,j}$ with a so-called pseudocount hyperparameter $\alpha \in \mathbb{R}_+$, giving

$$\varphi_{y,j} = \frac{\alpha + c_{v,j}}{V\alpha + \sum_{j=1}^{V} c_{y,j}}$$

where V is the number of features and $c_{y,j}$ is the observed count of the (y, j) label/feature pair. This technique introduces *bias*—it moves our predictions away from the optimal ones obtained with maximum likelihood estimation—but reduces *variance*.

2.2.5 Setting Hyperparameters

This subsection describes how one might go about setting model hyperparameters—like α—in an unbiased fashion.

2.3 Discriminative Learning

Naïve Bayes is an instance of a *generative model*; it estimating the joint probability of labels and features $p(x, y)$ and as such requires us to model the probability of the (features of) the text x. In contrast *discriminative models* directly focus on predicting $\hat{y}$.

2.3.1 Perceptron

The *perceptron* is a simple, online, discriminative learning method for linear classifiers. The perceptron learning algorithm is guaranteed to converge in a finite number of iterations when the data is *linearly separable*. Weights are initialized as zeros. During training, one iterates over examples—often in a random order created by shuffling them—and then computes $\hat{y} = \arg\max_{y \in \mathcal{Y}} \theta^{(t-1)} \cdot f(x^{(i)}, y)$ (algorithm 3, line 7). if $\hat{y} = y^{(i)}$; i.e., if this is the correct label, then the weights are unchanged (line 11). However, if not, we set them to $\theta^{(t-1)} + f(x^{(i)}, y^{(i)}) - f(x^{(i)}, \hat{y})$. This essentially increases the score of the correct label and decreases that of the incorrect label.

2.3.2 Averaged Perceptron

To achieve faster convergence—and to prevent problems in the case that the data is not linearly separable—one can average the weights across time, giving rise to the *averaged perceptron*. During training, we use the unaveraged weights, but use the averaged weights for inference afterwards.

2.4 Loss Functions and Large-Margin Classification

This section introduces the notion of *loss*, a data-specific measurement of classification performance. When there is no closed form solution to an optimization problem, as is the case for perceptrons, we can proceed by minimizing the loss.

2.4.1 Online Large-Margin Classification

Large-margin classification methods attempt to not merely classify each example correctly, but do so with a substantial wiggle room. Recall that the score for a correct label is given by $\theta \cdot f(x^{(i)}, y^{(i)})$, and let the score of the highest-scoring incorrect label be $\max_{y \neq y^{(i)}} \theta \cdot f(x^{(i)}, y)$. Then, the *margin* is defined as the difference of the two, is defined as

$$\gamma(\theta; x^(i), y^(i)) = \theta \cdot f(x^{(i)}, y^{(i)}) - \max_{y \neq y^{(i)}} \theta \cdot f(x^{(i)}, y)$$

The intuition here is that when the margin is large (and positive), the correct answer is separated from the next-closest incorrect answer by a large factor. The *online support vector machine* (or SVM) is an example of a linear classifier which attempts to maximize the margin.

2.4.2 Derivation of the Online Support Vector Machine

This subsection derives an online method for SVMs; the method described requires us to specify a hyperparameter $C \in \mathbb{R}_+$ controlling the bias-variance tradeoff.

2.5 Logistic Regression

Logistic regression is another discriminative learning method that, unlike SVMs, allow for a probabilistic interpretation, because it estimates the conditional probability $p_{Y|X}$.

2.5.1 Regularization

Regularization techniques are used to improve model generalization (i.e., to avoid overfitting) of a machine learning model. Normally such methods are implementing by adding a (scalar) regularization penalty to the loss, a value derived by some function to θ. L_2 regularization, called because the scalar penalty is given by $\frac{\lambda}{2}\|\theta\|_2^2 = \frac{\lambda}{2}\sum_i \theta_i^2$ where $\lambda \in (0, 1)$ is a hyperparameter. One other common methods, discussed briefly in §2.7.1, are L_1 regularization, which uses $\frac{\lambda}{2}\|\theta\|_1^2 = \frac{\lambda}{2}\sum_i \theta_i$. While Eistenstein does not discuss it, one can use L_1 and L_2 regularization simultaneously, a method known as *elastic net* regularization.

2.5.2 Gradients

This subsection derives the gradient for logistic regression. The formula is defined by the expected feature counts under the present module minus the observed feature counts (cf. Eistenstein's eq. 2.65.):

$$E_{Y|X}[f(x^{(i)}, y)] - f(x^{(i)}, y)$$

2.6 Optimization

Each one of the models discussed so far corresponds to an optimization problem in which the goal is to find parameters θ which minimize or maximize the value of some function of θ. In Naïve Bayes, for instance, we maximize the joint likelihood of features x and observations y:

$$\log p(x^{(1:N)}, y^{(1:N)})$$

For this, there is a closed-form solution, namely maximum likelihood estimation. For SVMs, we minimize the margin (with optional regularization). For logistic regression, we minimize the negative log-likelihood (with optional regularization). For the latter two, there is no closed-form solution (e.g., we can't just count some properties in the training data), but the objective is convex, allowing us to use generic convex optimization routines.

2.6.1 Batch Optimization

Batch optimization describes optimization methods that process the entire training set at once; naturally such methods are only feasible when the training set fits into available memory. Let $\theta^{(t)}$ be the weights at time t, Let $\eta^{(t)} \in \mathbb{R}_+$ (*eta*) be the *learning rate* at time t, and let $\nabla_\theta L$ be the gradient of the loss function computed over the entire training set. Then,

$$\theta^{(t+1)} \to \theta^{(t)} - \eta^{(t)} \nabla_\theta L.$$

Note that η is conditioned on time t because we often gradually reduce the learning rate as training proceeds; this is required for provable convergence when objective is convex.

2.6.2 Online Optimization

In *online optimization* updates to the parameters are made using subsamples of the training data, possibly as small as a single example, as in classic perceptron and *stochastic gradient descent* learning, but more often a *minibatch* consisting of dozens or hundreds of examples. This requires an algorithm for constructing (i.e., sampling) batches, but otherwise can proceed as above, using each minibatch to compute $\nabla_\theta L$. Online optimization often gives rise to lazy updating strategies.

2.7 Additional Topics in Classification

2.7.1 Feature Selection by Regularization

L_1 regularization, which penalizes non-zero weights according to their magnitude, can be used for feature selection. In all the models discussed above, if a features's weight is zero, it can be safely omitted from subsequent computations. In contrast, L_2 regularization is much less likely to induce model sparsity.

2.7.2 Other Views of Logistic Regression

There are two other ways to understand logistic regression:

- an instance of a *generalized linear model*, such as those used for statistical inference
- a *maximum entropy* (or *maxent*) model.

2.8 Summary of Learning Algorithms

Four general methods have been described:

- **Naïve Bayes**: probabilistic, implementation is easy, but performance is poor with correlated features.
- **Perceptron**: implementation is easy, but non-probabilistic.

- **Support vector machine**: error/margin-driven approach gives best general results, but requires a black-box optimization procedure and non-probabilistic.
- **Logistic regression**: probabilistic, but requires a black-box optimization procedure.

From my perspective, naïve Bayes is no longer appropriate except as a pedagogical tool; the perceptron is great for huge problems with massive numbers of sparse features. For everything else, use SVMs—Kummerfeld et al. (2015), for example, find large-margin methods outperform logistic regression in a wide variety of NLP tasks—unless a probabilistic interpretation is required. Tools like Scikit-learn make it very easy to try multiple methods with minimal fuss.

References

Eistenstein, Jacob. 2019. *Introduction to Natural Language Processing*. MIT Press.

Kummerfeld, Jonathan K., Taylor Berg-Kirkpatrick, and Dan Klein. 2015. An empirical analysis of optimization for max-margin NLP. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 273–279.