Lecture 11 — 04/15/24
Generative Classifiers (Reference Sheet)[1]

1 Introduction

Machine learning (ML) is the study of algorithms that can "learn" to perform tasks without explicit instruction. ML plays many roles in linguistics, and in speech & language technology, it is most commonly used to ***classify*** (i.e., label) ambiguous linguistic signals or events. In ***classification*** problems, we are given some observation x and asked to predict an associated ***class*** (or ***label***) $\hat{y} \in \mathcal{Y}$ where $\mathcal{Y}$ is a finite set. For example, x could be a newspaper article and $\hat{y}$ could be its predicted genre, x could be a tweet and $\hat{y}$ could predict whether it is hate speech, or perhaps x could be a word and $\hat{y}$ could be its predicted part of speech.

Supervision It is occasionally possible to make useful predictions of this sort in an ***unsupervised*** scenario. But most applications use ***supervision***, that is, "gold" x, y pairs produced by human annotators and used to train the classification algorithm.[2] During training, we attempt to learn a ***decision function*** $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$; that is, a function from observations to classes.

1.1 Classes of classifiers

Breiman (2001) describes "two cultures"[3] in the use of statistical modeling:

- Under one culture we posit a stochastic model which is responsible for the observed data.
- The other "culture" assumes that the stochastic process responsible for the observed data is complex and unknowable (a black box), and rather than attempting to model the distribution itself, attempts to learn the decision function directly.

A related, somewhat more formal, distinction is proposed by Ng and Jordan (2002):

- ***Generative classifiers*** compute a joint probability distribution $P(x, y)$, using $P(y \mid x)$ as computed by Bayes' rule:

$$\mathcal{R} = \arg\max_{y} P(y \mid x) \tag{1}$$

- ***Discriminative classifiers*** either directly estimate the conditional distribution $P(y \mid x)$, or induce $\mathcal{R}$ directly, ignoring both the joint and conditional distributions.

Discriminative classifiers tend to have lower theoretical error bounds, and may be preferable on the basis of simplicity:

> ...one should solve the problem directly and never solve a more general problem as an intermediate step. (Vapnik 1998:12)

Software note

The Python library Scikit-learn (`sklearn`; Pedregosa et al. 2011 — `https://scikit-learn.org/`) is a comprehensive collection of carefully implemented machine learning methods including naïve Bayes, logistic regression, and perceptron classifiers. In NLTK (Bird, Klein, & Loper 2009), the module `nltk.classify` — `https://www.nltk.org/api/nltk.classify.html` — also has implementations of several types of classifier. In many cases though, it is necessary or desirable to roll your own.

2 Features

In classification problems, we rarely define probability distributions or decision functions using raw observations $x \in \mathcal{X}$ directly. Instead we define some ***feature extraction*** function $\Phi \subseteq \mathcal{X} \times \mathcal{D}$. This function takes observations in $\mathcal{X}$ as inputs and maps them onto a ***feature space***, $\mathcal{D}$. Designing feature functions for a given classification problem is known as ***feature engineering***.[4]

[1]Notation used here has been adapted from Ng and Jordan (2002) and previous iterations put together by Kyle Gorman

[2]For instance, unsupervised part-of-speech tagging has been studied for 25 years (Merialdo 1994), but the best results are quite a bit worse than can be obtained with as little as two hours of human annotation (Garrette & Baldridge 2013).

[3]Breiman uses the terms "data modeling culture" and "algorithmic modeling culture" for these two enterprises respectively, though I find those names misleading in the current context

[4]Feature engineering is more of an art than a science, requiring both domain knowledge and practice. One of the major insights motivating neural networks is the belief that their hidden layers "learn" complex and expressive feature extraction functions given trivial input features.

2.1 Binary encoding

We assume, without loss of generality, that the feature space consists of binary vectors of length n; that is, $\mathcal{D} = \{0, 1\}^n$. For certain types of classifiers, features can also be multinomial categorical (including ordinal) and numerical variables drawn from some known distribution, like a Gaussian or a Poisson distribution. In other cases, we encode these variables using binary features.

Multinomial variables Categorical variables which have $n > 2$ unique values can be encoded as a sequence of $n - 1$ binary variables. The most common strategy is known as ***one-hot*** (or ***dummy***, or ***treatment***) encoding. For example, to represent the casing of a token—an important feature for part-of-speech tagging or named entity recognition—we might use a categorical variable $X = \{\text{dc}, \text{lower}, \text{mixed}, \text{title}, \text{upper}\}$. We then can exactly encode this using $\{0, 1\}^4$ like so:

$$\begin{aligned}
\text{dc} &\rightarrow [0, 0, 0, 0] \\
\text{lower} &\rightarrow [1, 0, 0, 0] \\
\text{mixed} &\rightarrow [0, 1, 0, 0] \\
\text{title} &\rightarrow [0, 0, 1, 0] \\
\text{upper} &\rightarrow [0, 0, 0, 1]
\end{aligned}$$

Numerical variables Numerical (i.e., integral or continuous variables) can also be approximately encoded as a sequence of $n - 1$ binary variables after grouping the values of the variable into n bins. While there are many ways to bin numerical variables, one of the simplest is to partition the variable into n empirical quantiles and then one-hot encode the quantile labels using $n - 1$ binary variables. For instance, quartiles can be encoded using $\{0, 1\}^3$:

$$\begin{aligned}
\text{1st quartile} &\rightarrow [0, 0, 0] \\
\text{2st quartile} &\rightarrow [1, 0, 0] \\
\text{3rd quartile} &\rightarrow [0, 1, 0] \\
\text{4th quartile} &\rightarrow [0, 0, 1]
\end{aligned}$$

2.2 Feature sparsity

A feature vector (or feature function) is said to be ***sparse*** when the vast majority of values in the vector are 0. In such cases, it is often more efficient to store only those feature values which are non-zero.

3 Generative classification: the naïve Bayes classifier

For simplicity, consider a supervised binary classification task. Let each observation be a $x \in \mathcal{X}, y \in \mathcal{Y}$ pair where $\mathcal{Y} = \{0, 1\}$. Given a feature extraction function $\Phi \subseteq \mathcal{X} \times \{0, 1\}^n$, let $\mathcal{F} = \Phi(x)$. Then, let S denote a finite training set drawn from $\mathcal{F} \times \mathcal{Y}$.

The simplest generative classifier is the naïve Bayes classifier. Despite its simplicity and the naïveity of its underlying assumptions, it is said to be "unreasonably effective" (Zhang 2004) at simple problems.

3.1 Parameters

The parameters of this model are given by two probability distributions. The first is the maximum likelihood estimate (MLE) of the probability that $y = 1$, or

$$\hat{P}(y = 1) = \frac{C(y = 1)}{|S|} \tag{2}$$

where C gives the count of some event in S and $|S|$ is the size of the training set. The second is the MLE conditional probability of each (true) feature value given that $y = 1$, or

$$\hat{P}(\mathcal{F}_i = 1 \mid y = 1) = \frac{C(\mathcal{F}_i = 1, y = 1)}{C(y = 1)}. \tag{3}$$

Sometimes, we make use of add-α smoothing for this conditional probability distribution, so that

$$\hat{P}(\mathcal{F}_i = 1 \mid y = 1) = \frac{C(\mathcal{F}_i = 1, y = 1) + \alpha}{C(y = 1) + 2\alpha}. \tag{4}$$

3.2 Decision rule

The naïve Bayes model is called "naïve" because it assumes that the probability of each feature value $\mathcal{F}_i$ is conditionally independent of every other feature $\mathcal{F}_{j \neq i}$, or

$$P(\mathcal{F}_i = 1 \mid \mathcal{F}_{j \neq i} = 1, \ldots, y = 1) = P(\mathcal{F}_i = 1 \mid y = 1). \tag{5}$$

Under this assumption,

$$\hat{P}(y = 1 \mid \mathcal{F}) = \hat{P}(y = 1) \prod_{i=1}^{n} \hat{P}(\mathcal{F}_i = 1 \mid y = 1). \tag{6}$$

We can write the ***posterior probability*** of some class $y' \in \mathcal{Y}$ as:

$$\hat{P}(y = y' \mid \mathcal{F}) = \hat{P}(y = y') \prod_{i=1}^{n} \hat{P}(\mathcal{F}_i = 1 \mid y = y'). \tag{7}$$

Then, to classify a feature vector $\mathcal{F}$, we wish to pick the most probable class $\hat{y}$ according to

$$\hat{y} = \operatorname*{arg\,max}_{y' \in \mathcal{Y}} \hat{P}(y = y' \mid \mathcal{F}) \tag{8}$$

This is the ***maximum a posteriori*** (or MAP) decision rule. In the binary case, this simplifies as follows (thanks to the law of total probability):

$$\hat{y} = \begin{cases} 1 \text{ if } \hat{P}(y = 1 \mid \mathcal{F}) > .5 \\ 0 \text{ otherwise} \end{cases} \tag{9}$$

3.3 Avoiding underflow

As is often the case when computing the product of probabilities, there is a risk of underflow. We can avoid this by storing $\log \hat{P}(y = 1)$ and $\log \hat{P}(\mathcal{F}_i \mid y = 1)$. The log posterior probability is

$$\log \hat{P}(y = y' \mid \mathcal{F}) = \log \hat{P}(y = y') + \sum_{i=1}^{n} \log \hat{P}(\mathcal{F}_i = 1 \mid y = y') \tag{10}$$

and the decision rule is

$$\hat{y} = \operatorname*{arg\,max}_{y' \in \mathcal{Y}} \left[\log \hat{P}(y = y' \mid \mathcal{F}) \right]. \tag{11}$$

3.4 Multinomial classification

It is straightforward to adapt this model for cases where $|\mathcal{Y}| > 2$. To apply the decision rule (eq. 8), one needs merely to compute $\hat{P}(y = y')$ and $\hat{P}(\mathcal{F}_i = 1 \mid y = y')$ for all $y' \in \mathcal{Y}$, not just $y = 1$, and select whichever y' gives the highest value to the posterior probability distribution.

References

Bird, S., Klein, E., & Loper, E. (2009). ***Natural language processing with python***. O'Reilly Media.

Breiman, L. (2001). Statistical modeling: the two cultures. ***Statistical Science***, ***16***(3), 199-231.

Garrette, D., & Baldridge, J. (2013). Learning a part-of-speech tagger from two hours of annotation. In ***Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*** (p. 138-147).

Merialdo, B. (1994). Tagging English text with a probabilistic model. ***Computational Linguistics***, ***20***(2), 155-171.

Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. generative classifiers: a comparison of logistic regression and naive Bayes. In ***Proceedings of neurips*** (p. 841-848).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: machine learning in Python. ***Journal of Machine Learning Research***, ***12***, 2825-2830.

Vapnik, V. N. (1998). ***Statistical learning theory***. Wiley.

Zhang, H. (2004). The optimality of naive Bayes. In ***Seventeenth international florida artificial intelligence research society conference*** (p. 562-567).