1.9. Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable y and dependent feature vector x_1 through x_n, :

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)}$$

Using the naive conditional independence assumption that

$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y),$$

for all i, this relationship is simplified to

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

$$\begin{aligned} P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \\ \Downarrow \\ \hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y), \end{aligned}$$

and we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$; the former is then the relative frequency of class y in the training set.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$.

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters. (For theoretical reasons why naive Bayes works well, and on which types of data it does, see the references below.)

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator, so the probability outputs from `predict_proba` are not to be taken too seriously.

References:

- H. Zhang (2004). The optimality of Naive Bayes. Proc. FLAIRS.

1.9.1. Gaussian Naive Bayes

`GaussianNB` implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma^2_y}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma^2_y}\right)$$

The parameters σ_y and μ_y are estimated using maximum likelihood.

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.naive_bayes import GaussianNB
>>> X, y = load_iris(return_X_y=True)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
>>> gnb = GaussianNB()
>>> y_pred = gnb.fit(X_train, y_train).predict(X_test)
>>> print("Number of mislabeled points out of a total %d points : %d"
...       % (X_test.shape[0], (y_test != y_pred).sum()))
Number of mislabeled points out of a total 75 points : 4
```

1.9.2. Multinomial Naive Bayes

`MultinomialNB` implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors $\theta_y = (\theta_{y1},\ldots,\theta_{yn})$ for each class y, where n is the number of features (in text classification, the size of the vocabulary) and θ_{yi} is the probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y.

The parameters θ_y is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{ N_{yi} + \alpha}{N_y + \alpha n}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and $N_{y} = \sum_{i=1}^{n} N_{yi}$ is the total count of all features for class y.

The smoothing priors $\alpha \ge 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha = 1$ is called Laplace smoothing, while $\alpha < 1$ is called Lidstone smoothing.

1.9.3. Complement Naive Bayes

`ComplementNB` implements the complement naive Bayes (CNB) algorithm. CNB is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets. Specifically, CNB uses statistics from the *complement* of each class to compute the model's weights. The inventors of CNB show empirically that the parameter estimates for CNB are more stable than those for MNB. Further, CNB regularly outperforms MNB (often by a considerable margin) on text classification tasks. The procedure for calculating the weights is as follows:

$$\begin{aligned} \hat{\theta}_{ci} = \frac{\alpha_i + \sum_{j:y_j \neq c} d_{ij}}{\alpha + \sum_{j:y_j \neq c} \sum_{k} d_{kj}} \\ w_{ci} = \log \hat{\theta}_{ci} \\ w_{ci} = \frac{w_{ci}}{\sum_{j} |w_{cj}|} \end{aligned}$$

where the summations are over all documents j not in class c, d_{ij} is either the count or tf-idf value of term i in document j, α_i is a smoothing hyperparameter like that found in MNB, and $\alpha = \sum_{i} \alpha_i$. The second normalization addresses the tendency for longer documents to dominate parameter estimates in MNB. The classification rule is:

$$\hat{c} = \arg\min_c \sum_{i} t_i w_{ci}$$

i.e., a document is assigned to the class that is the *poorest* complement match.

References:

- Rennie, J. D., Shih, L., Teevan, J., & Karger, D. R. (2003). Tackling the poor assumptions of naive bayes text classifiers. In ICML (Vol. 3, pp. 616-623).

1.9.4. Bernoulli Naive Bayes

`BernoulliNB` implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable. Therefore, this class requires samples to be represented as binary-valued feature vectors; if handed any other kind of data, a `BernoulliNB` instance may binarize its input (depending on the `binarize` parameter).

The decision rule for Bernoulli naive Bayes is based on

$$P(x_i \mid y) = P(x_i = 1 \mid y) x_i + (1 - P(x_i = 1 \mid y)) (1 - x_i)$$

which differs from multinomial NB's rule in that it explicitly penalizes the non-occurrence of a feature i that is an indicator for class y, where the multinomial variant would simply ignore a non-occurring feature.

In the case of text classification, word occurrence vectors (rather than word count vectors) may be used to train and use this classifier. `BernoulliNB` might perform better on some datasets, especially those with shorter documents. It is advisable to evaluate both models, if time permits.

References:

- C.D. Manning, P. Raghavan and H. Schütze (2008). Introduction to Information Retrieval. Cambridge University Press, pp. 234-265.
- A. McCallum and K. Nigam (1998). A comparison of event models for Naive Bayes text classification. Proc. AAAI/ICML-98 Workshop on Learning for Text Categorization, pp. 41-48.
- V. Metsis, I. Androutsopoulos and G. Paliouras (2006). Spam filtering with Naive Bayes – Which Naive Bayes? 3rd Conf. on Email and Anti-Spam (CEAS).

1.9.5. Categorical Naive Bayes

`CategoricalNB` implements the categorical naive Bayes algorithm for categorically distributed data. It assumes that each feature, which is described by the index i, has its own categorical distribution.

For each feature i in the training set X, `CategoricalNB` estimates a categorical distribution for each feature i of X conditioned on the class y. The index set of the samples is defined as $J = \{ 1, \dots, m \}$, with m as the number of samples.

The probability of category t in feature i given class c is estimated as:

$$P(x_i = t \mid y = c \: ;\, \alpha) = \frac{ N_{tic} + \alpha}{N_{c} + \alpha n_i},$$

where $N_{tic} = |\{j \in J \mid x_{ij} = t, y_j = c\}|$ is the number of times category t appears in the samples x_{i}, which belong to class c, $N_{c} = |\{ j \in J\mid y_j = c\}|$ is the number of samples with class c, α is a smoothing parameter and n_i is the number of available categories of feature i.

`CategoricalNB` assumes that the sample matrix X is encoded (for instance with the help of `OrdinalEncoder`) such that all categories for each feature i are represented with numbers $0, ..., n_i - 1$ where n_i is the number of available categories of feature i.

1.9.6. Out-of-core naive Bayes model fitting

Naive Bayes models can be used to tackle large scale classification problems for which the full training set might not fit in memory. To handle this case, `MultinomialNB`, `BernoulliNB`, and `GaussianNB` expose a `partial_fit` method that can be used incrementally as done with other classifiers as demonstrated in Out-of-core classification of text documents. All naive Bayes classifiers support sample weighting.

Contrary to the `fit` method, the first call to `partial_fit` needs to be passed the list of all the expected class labels.

For an overview of available strategies in scikit-learn, see also the out-of-core learning documentation.

Note: The `partial_fit` method call of naive Bayes models introduces some computational overhead. It is recommended to use data chunk sizes that are as large as possible, that is as the available RAM allows.

