Multinomial Naive Bayes

A multinomial distribution is useful to model feature vectors where each value represents, for example, the number of occurrences of a term or its relative frequency. If the feature vectors have *n* elements and each of them can assume *k* different values with probability p_k, then:

$$p(X_1 = x_1 \cap X_2 = x_2 \ \cap \ \ldots \cap X_k = x_k) = \frac{n!}{\prod_i x_i!} \prod_j p_j^{x_j}$$

The conditional probabilities $P(x^{(i)}|y_j)$ are computed with a frequency count (which corresponds to applying a maximum likelihood approach), but in this case, it's important to consider also a correction parameter α (called **Laplace** or **Lidstone smoothing factor**), to avoid null probabilities:

$$p(\bar{x}^{(i)}|y_j) = \frac{(N_{\bar{x}^{(i)}} = y_j) + \alpha}{N(y_j) + m\alpha}$$

In the previous expression, the numerator is analogous to the Bernoulli case (with the addition of α), in fact, it counts the number of occurrences of the y_j class in the i^{th} feature of the input samples. The denominator, instead, is the total count of occurrences of the y_j class (in all features) plus a correction factor which is proportional to the dimensionality of the inputs. In case of null counts, the probability defaults on a constant value. This correction is very helpful when evaluating samples whose features are not included in the training set. Even if this is a situation that should be properly managed, without the correction, their probability would be null. For example, in NLP, it's common to build a dictionary starting from a corpus of documents and then splitting a vectorized dataset (where each element contains n features representing the number of occurrences—or a function of it—of a specific word). If a word doesn't appear in any document belonging to the training set, a non-smoothed model wouldn't be able to properly manage the document in the majority of cases (that is, considering only the known elements to assign the most likely class).

The default value for *α* is *1.0* (in this case, it's called Laplace factor) and it prevents the model from setting null probabilities when the frequency is zero. It's possible to assign all non-negative values; however, larger values will assign higher probabilities to the missing features, and this choice could alter the stability of the model. When $\alpha < 1.0$, it's usually called the **Lidstone factor**. Clearly, if $\alpha \rightarrow 0$, the effect becomes more and more negligible, returning to a scenario very similar to the Bernoulli Naive Bayes. In our example, we're going to consider the default value of *1.0*.

For our purposes, we're going to use `DictVectorizer`, already analyzed in Chapter 2, *Important Elements in Machine Learning*. There are automatic instruments to compute the frequencies of terms, but we're going to discuss them later. Let's consider only two records: the first one representing a city and the second one the countryside. Our dictionary contains hypothetical frequencies as if the terms were extracted from a text description:

```
from sklearn.feature_extraction import DictVectorizer

data = [
   {'house': 100, 'street': 50, 'shop': 25, 'car': 100, 'tree': 20},
   {'house': 5, 'street': 5, 'shop': 0, 'car': 10, 'tree': 500, 'river': 1}
]

dv = DictVectorizer(sparse=False)
X = dv.fit_transform(data)
Y = np.array([1, 0])

print(X)
[[ 100.,  100.,    0.,   25.,   50.,   20.],
 [  10.,    5.,    1.,    0.,    5.,  500.]]
```

Note that the term `'river'` is missing from the first set, so it's useful to keep α equal to `1.0` to give it a small probability. The output classes are `1` for city and `0` for the countryside. Now we can train a `MultinomialNB` instance:

```
from sklearn.naive_bayes import MultinomialNB

mnb = MultinomialNB()
mnb.fit(X, Y)
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

To test the model, we create a dummy city with a `river` and a dummy countryside place without any `river`:

```
test_data = data = [
    {'house': 80, 'street': 20, 'shop': 15, 'car': 70, 'tree': 10, 'river':
    {'house': 10, 'street': 5, 'shop': 1, 'car': 8, 'tree': 300, 'river': 0}
]

print(mnb.predict(dv.fit_transform(test_data)))
[1, 0]
```

As expected, the prediction is correct. Later on, when discussing some elements of NLP, we're going to use a **Multinomial Naive Bayes** for text classification with larger corpora. Even if a multinomial distribution is based on the number of occurrences, it can be used successfully with frequencies or more complex functions.