Token	1	2	3	4	5	6	7	8	9	10
Word	mary	lamb	little	big	fleece	white	black	snow	rain	unk
Count	2	4	4	0	1	1	0	1	0	4

Denote the above counts by N_j. If we use a $\text{Dir}(\boldsymbol{\alpha})$ prior for $\boldsymbol{\theta}$, the posterior predictive is just

$$p(\tilde{X} = j|D) = E[\theta_j|D] = \frac{\alpha_j + N_j}{\sum_{j'} \alpha_{j'} + N_{j'}} = \frac{1 + N_j}{10 + 17} \tag{3.52}$$

If we set $\alpha_j = 1$, we get

$$p(\tilde{X} = j|D) = (3/27, 5/27, 5/27, 1/27, 2/27, 2/27, 1/27, 2/27, 1/27, 5/27) \tag{3.53}$$

The modes of the predictive distribution are $X = 2$ ("lamb") and $X = 10$ ("unk"). Note that the words "big", "black" and "rain" are predicted to occur with non-zero probability in the future, even though they have never been seen before. Later on we will see more sophisticated language models.

3.5 Naive Bayes classifiers

In this section, we discuss how to classify vectors of discrete-valued features, $\mathbf{x} \in \{1, \ldots, K\}^D$, where K is the number of values for each feature, and D is the number of features. We will use a generative approach. This requires us to specify the class conditional distribution, $p(\mathbf{x}|y = c)$. The simplest approach is to assume the features are **conditionally independent** given the class label. This allows us to write the class conditional density as a product of one dimensional densities:

$$p(\mathbf{x}|y = c, \boldsymbol{\theta}) = \prod_{j=1}^{D} p(x_j|y = c, \boldsymbol{\theta}_{jc}) \tag{3.54}$$

The resulting model is called a **naive Bayes classifier** (NBC).

The model is called "naive" since we do not expect the features to be independent, even conditional on the class label. However, even if the naive Bayes assumption is not true, it often results in classifiers that work well (Domingos and Pazzani 1997). One reason for this is that the model is quite simple (it only has $O(CD)$ parameters, for C classes and D features), and hence it is relatively immune to overfitting.

The form of the class-conditional density depends on the type of each feature. We give some possibilities below:

- In the case of real-valued features, we can use the Gaussian distribution: $p(\mathbf{x}|y = c, \boldsymbol{\theta}) = \prod_{j=1}^{D} \mathcal{N}(x_j|\mu_{jc}, \sigma_{jc}^2)$, where μ_{jc} is the mean of feature j in objects of class c, and σ_{jc}^2 is its variance.
- In the case of binary features, $x_j \in \{0, 1\}$, we can use the Bernoulli distribution: $p(\mathbf{x}|y = c, \boldsymbol{\theta}) = \prod_{j=1}^{D} \text{Ber}(x_j|\mu_{jc})$, where μ_{jc} is the probability that feature j occurs in class c. This is sometimes called the **multivariate Bernoulli naive Bayes** model. We will see an application of this below.

- In the case of categorical features, $x_j \in \{1, \ldots, K\}$, we can model use the multinoulli distribution: $p(\mathbf{x}|y = c, \boldsymbol{\theta}) = \prod_{j=1}^{D} \mathrm{Cat}(x_j|\boldsymbol{\mu}_{jc})$, where $\boldsymbol{\mu}_{jc}$ is a histogram over the K possible values for x_j in class c.

Obviously we can handle other kinds of features, or use different distributional assumptions. Also, it is easy to mix and match features of different types.

3.5.1 Model fitting

We now discuss how to "train" a naive Bayes classifier. This usually means computing the MLE or the MAP estimate for the parameters. However, we will also discuss how to compute the full posterior, $p(\boldsymbol{\theta}|\mathcal{D})$.

3.5.1.1 MLE for NBC

The probability for a single data case is given by

$$p(\mathbf{x}_i, y_i|\boldsymbol{\theta}) = p(y_i|\boldsymbol{\pi}) \prod_j p(x_{ij}|\boldsymbol{\theta}_j) = \prod_c \pi_c^{\mathbb{I}(y_i=c)} \prod_j \prod_c p(x_{ij}|\boldsymbol{\theta}_{jc})^{\mathbb{I}(y_i=c)} \tag{3.55}$$

Hence the log-likelihood is given by

$$\log p(\mathcal{D}|\boldsymbol{\theta}) \quad = \quad \sum_{c=1}^{C} N_c \log \pi_c + \sum_{j=1}^{D} \sum_{c=1}^{C} \sum_{i:y_i=c} \log p(x_{ij}|\boldsymbol{\theta}_{jc}) \tag{3.56}$$

We see that this expression decomposes into a series of terms, one concerning $\boldsymbol{\pi}$, and DC terms containing the $\boldsymbol{\theta}_{jc}$'s. Hence we can optimize all these parameters separately.

From Equation 3.48, the MLE for the class prior is given by

$$\hat{\pi}_c \quad = \quad \frac{N_c}{N} \tag{3.57}$$

where $N_c \triangleq \sum_i \mathbb{I}(y_i = c)$ is the number of examples in class c.

The MLE for the likelihood depends on the type of distribution we choose to use for each feature. For simplicity, let us suppose all features are binary, so $x_j|y = c \sim \mathrm{Ber}(\theta_{jc})$. In this case, the MLE becomes

$$\hat{\theta}_{jc} = \frac{N_{jc}}{N_c} \tag{3.58}$$

It is extremely simple to implement this model fitting procedure: See Algorithm 8 for some pseudo-code (and `naiveBayesFit` for some Matlab code). This algorithm obviously takes $O(ND)$ time. The method is easily generalized to handle features of mixed type. This simplicity is one reason the method is so widely used.

Figure 3.8 gives an example where we have 2 classes and 600 binary features, representing the presence or absence of words in a bag-of-words model. The plot visualizes the $\boldsymbol{\theta}_c$ vectors for the two classes. The big spike at index 107 corresponds to the word "subject", which occurs in both classes with probability 1. (In Section 3.5.4, we discuss how to "filter out" such uninformative features.)

Algorithm 3.1: Fitting a naive Bayes classifier to binary features

$N_c = 0$, $N_{jc} = 0$;
for $i = 1 : N$ **do**
  $c = y_i$ // Class label of i'th example;
  $N_c := N_c + 1$;
  for $j = 1 : D$ **do**
    if $x_{ij} = 1$ **then**
      $N_{jc} := N_{jc} + 1$
$\hat{\pi}_c = \frac{N_c}{N}$, $\hat{\theta}_{jc} = \frac{N_{jc}}{N}$

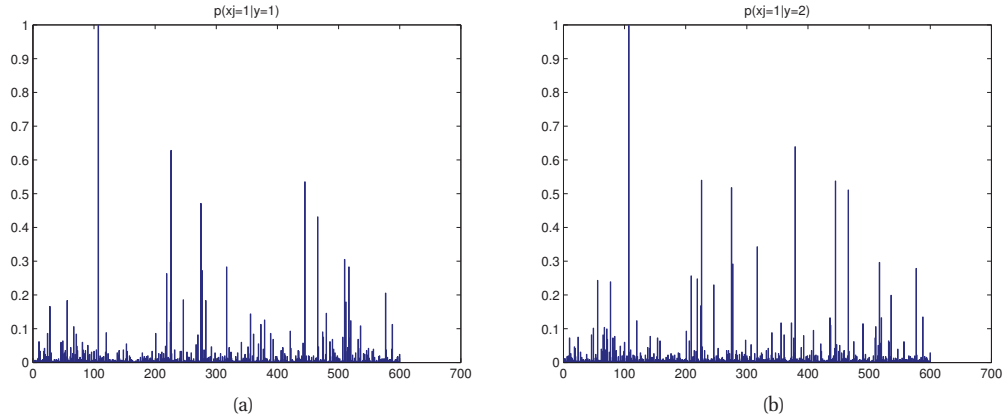

Figure 3.8 Class conditional densities $p(x_j = 1|y = c)$ for two document classes, corresponding to "X windows" and "MS windows". Figure generated by `naiveBayesBowDemo`.

3.5.1.2 Bayesian naive Bayes

The trouble with maximum likelihood is that it can overfit. For example, consider the example in Figure 3.8: the feature corresponding to the word "subject" (call it feature j) always occurs in both classes, so we estimate $\hat{\theta}_{jc} = 1$. What will happen if we encounter a new email which does not have this word in it? Our algorithm will crash and burn, since we will find that $p(y = c|\mathbf{x}, \hat{\boldsymbol{\theta}}) = 0$ for both classes! This is another manifestation of the black swan paradox discussed in Section 3.3.4.1.

A simple solution to overfitting is to be Bayesian. For simplicity, we will use a factored prior:

$$p(\boldsymbol{\theta}) \quad = \quad p(\boldsymbol{\pi}) \prod_{j=1}^{D} \prod_{c=1}^{C} p(\theta_{jc}) \tag{3.59}$$

We will use a $\mathrm{Dir}(\boldsymbol{\alpha})$ prior for $\boldsymbol{\pi}$ and a $\mathrm{Beta}(\beta_0, \beta_1)$ prior for each θ_{jc}. Often we just take $\boldsymbol{\alpha} = \mathbf{1}$ and $\boldsymbol{\beta} = \mathbf{1}$, corresponding to add-one or Laplace smoothing.

Combining the factored likelihood in Equation 3.56 with the factored prior above gives the following factored posterior:

$$p(\boldsymbol{\theta}|\mathcal{D}) = p(\boldsymbol{\pi}|\mathcal{D}) \prod_{j=1}^{D} \prod_{c=}^{C} p(\theta_{jc}|\mathcal{D}) \tag{3.60}$$

$$p(\boldsymbol{\pi}|\mathcal{D}) = \mathrm{Dir}(N_1 + \alpha_1 \ldots, N_C + \alpha_C) \tag{3.61}$$

$$p(\theta_{jc}|\mathcal{D}) = \mathrm{Beta}((N_c - N_{jc}) + \beta_0, N_{jc} + \beta_1) \tag{3.62}$$

In other words, to compute the posterior, we just update the prior counts with the empirical counts from the likelihood. It is straightforward to modify algorithm 8 to handle this version of model "fitting".

3.5.2 Using the model for prediction

At test time, the goal is to compute

$$p(y = c|\mathbf{x}, \mathcal{D}) \propto p(y = c|\mathcal{D}) \prod_{j=1}^{D} p(x_j|y = c, \mathcal{D}) \tag{3.63}$$

The correct Bayesian procedure is to integrate out the unknown parameters:

$$p(y = c|\mathbf{x}, \mathcal{D}) \propto \left[\int \mathrm{Cat}(y = c|\boldsymbol{\pi}) p(\boldsymbol{\pi}|\mathcal{D}) d\boldsymbol{\pi} \right] \tag{3.64}$$

$$\prod_{j=1}^{D} \left[\int \mathrm{Ber}(x_j|y = c, \theta_{jc}) p(\boldsymbol{\theta}_{jc}|\mathcal{D}) \right] \tag{3.65}$$

Fortunately, this is easy to do, at least if the posterior is Dirichlet. In particular, from Equation 3.51, we know the posterior predictive density can be obtained by simply plugging in the posterior mean parameters $\overline{\boldsymbol{\theta}}$. Hence

$$p(y = c|\mathbf{x}, \mathcal{D}) \propto \overline{\pi}_c \prod_{j=1}^{D} (\overline{\theta}_{jc})^{\mathbb{I}(x_j=1)} (1 - \overline{\theta}_{jc})^{\mathbb{I}(x_j=0)} \tag{3.66}$$

$$\overline{\theta}_{jk} = \frac{N_{jc} + \beta_1}{N_c + \beta_0 + \beta_1} \tag{3.67}$$

$$\overline{\pi}_c = \frac{N_c + \alpha_c}{N + \alpha_0} \tag{3.68}$$

where $\alpha_0 = \sum_c \alpha_c$.

If we have approximated the posterior by a single point, $p(\boldsymbol{\theta}|\mathcal{D}) \approx \delta_{\hat{\boldsymbol{\theta}}}(\boldsymbol{\theta})$, where $\hat{\boldsymbol{\theta}}$ may be the ML or MAP estimate, then the posterior predictive density is obtained by simply plugging in the parameters, to yield a virtually identical rule:

$$p(y = c|\mathbf{x}, \mathcal{D}) \propto \hat{\pi}_c \prod_{j=1}^{D} (\hat{\theta}_{jc})^{\mathbb{I}(x_j=1)} (1 - \hat{\theta}_{jc})^{\mathbb{I}(x_j=0)} \tag{3.69}$$

The only difference is we replaced the posterior mean $\overline{\boldsymbol{\theta}}$ with the posterior mode or MLE $\hat{\boldsymbol{\theta}}$. However, this small difference can be important in practice, since the posterior mean will result in less overfitting (see Section 3.4.4.1).

3.5.3 The log-sum-exp trick

We now discuss one important practical detail that arises when using generative classifiers of any kind. We can compute the posterior over class labels using Equation 2.13, using the appropriate class-conditional density (and a plug-in approximation). Unfortunately a naive implementation of Equation 2.13 can fail due to **numerical underflow**. The problem is that $p(\mathbf{x}|y=c)$ is often a very small number, especially if $\mathbf{x}$ is a high-dimensional vector. This is because we require that $\sum_{\mathbf{x}} p(\mathbf{x}|y) = 1$, so the probability of observing any particular high-dimensional vector is small. The obvious solution is to take logs when applying Bayes rule, as follows:

$$\log p(y=c|\mathbf{x}) = b_c - \log\left[\sum_{c'=1}^{C} e^{b_{c'}}\right] \tag{3.70}$$

$$b_c \triangleq \log p(\mathbf{x}|y=c) + \log p(y=c) \tag{3.71}$$

However, this requires evaluating the following expression

$$\log[\sum_{c'} e^{b_{c'}}] = \log \sum_{c'} p(y=c', \mathbf{x}) = \log p(\mathbf{x}) \tag{3.72}$$

and we can't add up in the log domain. Fortunately, we can factor out the largest term, and just represent the remaining numbers relative to that. For example,

$$\log(e^{-120} + e^{-121}) = \log\left(e^{-120}(e^0 + e^{-1})\right) = \log(e^0 + e^{-1}) - 120 \tag{3.73}$$

In general, we have

$$\log \sum_c e^{b_c} = \log\left[(\sum_c e^{b_c - B})e^B\right] = \left[\log(\sum_c e^{b_c - B})\right] + B \tag{3.74}$$

where $B = \max_c b_c$. This is called the **log-sum-exp** trick, and is widely used. (See the function `logsumexp` for an implementation.)

This trick is used in Algorithm 1 which gives pseudo-code for using an NBC to compute $p(y_i|\mathbf{x}_i, \hat{\boldsymbol{\theta}})$. See `naiveBayesPredict` for the Matlab code. Note that we do not need the log-sum-exp trick if we only want to compute $\hat{y}_i$, since we can just maximize the unnormalized quantity $\log p(y_i = c) + \log p(\mathbf{x}_i|y=c)$.

3.5.4 Feature selection using mutual information

Since an NBC is fitting a joint distribution over potentially many features, it can suffer from overfitting. In addition, the run-time cost is $O(D)$, which may be too high for some applications.

One common approach to tackling both of these problems is to perform **feature selection**, to remove "irrelevant" features that do not help much with the classification problem. The simplest approach to feature selection is to evaluate the relevance of each feature separately, and then

Algorithm 3.2: Predicting with a naive bayes classifier for binary features

for $i = 1 : N$ **do**
  for $c = 1 : C$ **do**
    $L_{ic} = \log \hat{\pi}_c$;
    for $j = 1 : D$ **do**
      if $x_{ij} = 1$ **then** $L_{ic} := L_{ic} + \log \hat{\theta}_{jc}$ **else** $L_{ic} := L_{ic} + \log(1 - \hat{\theta}_{jc})$
  $p_{ic} = \exp(L_{ic} - \text{logsumexp}(L_{i,:}))$;
  $\hat{y}_i = \text{argmax}_c \, p_{ic}$;

take the top K, where K is chosen based on some tradeoff between accuracy and complexity. This approach is known as variable **ranking**, **filtering**, or **screening**.

One way to measure relevance is to use mutual information (Section 2.8.3) between feature X_j and the class label Y:

$$I(X, Y) = \sum_{x_j} \sum_{y} p(x_j, y) \log \frac{p(x_j, y)}{p(x_j)p(y)} \tag{3.75}$$

The mutual information can be thought of as the reduction in entropy on the label distribution once we observe the value of feature j. If the features are binary, it is easy to show (Exercise 3.21) that the MI can be computed as follows

$$I_j = \sum_{c} \left[\theta_{jc} \pi_c \log \frac{\theta_{jc}}{\theta_j} + (1 - \theta_{jc}) \pi_c \log \frac{1 - \theta_{jc}}{1 - \theta_j} \right] \tag{3.76}$$

where $\pi_c = p(y = c)$, $\theta_{jc} = p(x_j = 1 | y = c)$, and $\theta_j = p(x_j = 1) = \sum_c \pi_c \theta_{jc}$. (All of these quantities can be computed as a by-product of fitting a naive Bayes classifier.)

Figure 3.1 illustrates what happens if we apply this to the binary bag of words dataset used in Figure 3.8. We see that the words with highest mutual information are much more discriminative than the words which are most probable. For example, the most probable word in both classes is "subject", which always occurs because this is newsgroup data, which always has a subject line. But obviously this is not very discriminative. The words with highest MI with the class label are (in decreasing order) "windows", "microsoft", "DOS" and "motif", which makes sense, since the classes correspond to Microsoft Windows and X Windows.

3.5.5 Classifying documents using bag of words

Document classification is the problem of classifying text documents into different categories. One simple approach is to represent each document as a binary vector, which records whether each word is present or not, so $x_{ij} = 1$ iff word j occurs in document i, otherwise $x_{ij} = 0$. We can then use the following class conditional density:

$$p(\mathbf{x}_i | y_i = c, \boldsymbol{\theta}) = \prod_{j=1}^{D} \text{Ber}(x_{ij} | \theta_{jc}) = \prod_{j=1}^{D} \theta_{jc}^{\mathbb{I}(x_{ij})} (1 - \theta_{jc})^{\mathbb{I}(1 - x_{ij})} \tag{3.77}$$

class 1	prob	class 2	prob	highest MI	MI
subject	0.998	subject	0.998	windows	0.215
this	0.628	windows	0.639	microsoft	0.095
with	0.535	this	0.540	dos	0.092
but	0.471	with	0.538	motif	0.078
you	0.431	but	0.518	window	0.067

Table 3.1 We list the 5 most likely words for class 1 (X windows) and class 2 (MS windows). We also show the 5 words with highest mutual information with class label. Produced by `naiveBayesBowDemo`

This is called the **Bernoulli product model**, or the **binary independence model**.

However, ignoring the number of times each word occurs in a document loses some information (McCallum and Nigam 1998). A more accurate representation counts the number of occurrences of each word. Specifically, let $\mathbf{x}_i$ be a vector of counts for document i, so $x_{ij} \in \{0, 1, \ldots, N_i\}$, where N_i is the number of terms in document i (so $\sum_{j=1}^{D} x_{ij} = N_i$). For the class conditional densities, we can use a multinomial distribution:

$$p(\mathbf{x}_i|y_i = c, \boldsymbol{\theta}) = \text{Mu}(\mathbf{x}_i|N_i, \boldsymbol{\theta}_c) = \frac{N_i!}{\prod_{j=1}^{D} x_{ij}!} \prod_{j=1}^{D} \theta_{jc}^{x_{ij}} \tag{3.78}$$

where we have implicitly assumed that the document length N_i is independent of the class. Here θ_{jc} is the probability of generating word j in documents of class c; these parameters satisfy the constraint that $\sum_{j=1}^{D} \theta_{jc} = 1$ for each class c.[3]

Although the multinomial classifier is easy to train and easy to use at test time, it does not work particularly well for document classification. One reason for this is that it does not take into account the **burstiness** of word usage. This refers to the phenomenon that most words never appear in any given document, but if they do appear once, they are likely to appear more than once, i.e., words occur in bursts.

The multinomial model cannot capture the burstiness phenomenon. To see why, note that Equation 3.78 has the form $\theta_{jc}^{N_{ij}}$, and since $\theta_{jc} \ll 1$ for rare words, it becomes increasingly unlikely to generate many of them. For more frequent words, the decay rate is not as fast. To see why intuitively, note that the most frequent words are function words which are not specific to the class, such as "and", "the", and "but"; the chance of the word "and" occuring is pretty much the same no matter how many time it has previously occurred (modulo document length), so the independence assumption is more reasonable for common words. However, since rare words are the ones that matter most for classification purposes, these are the ones we want to model the most carefully.

Various ad hoc heuristics have been proposed to improve the performance of the multinomial document classifier (Rennie et al. 2003). We now present an alternative class conditional density that performs as well as these ad hoc methods, yet is probabilistically sound (Madsen et al. 2005).

3. Since Equation 3.78 models each word independently, this model is often called a naive Bayes classifier, although technically the features x_{ij} are not independent, because of the constraint $\sum_j x_{ij} = N_i$.

Suppose we simply replace the multinomial class conditional density with the **Dirichlet Compound Multinomial** or **DCM** density, defined as follows:

$$p(\mathbf{x}_i|y_i = c, \boldsymbol{\alpha}) = \int \mathrm{Mu}(\mathbf{x}_i|N_i, \boldsymbol{\theta}_c)\mathrm{Dir}(\boldsymbol{\theta}_c|\boldsymbol{\alpha}_c)d\boldsymbol{\theta}_c = \frac{N_i!}{\prod_{j=1}^{D} x_{ij}!}\frac{B(\mathbf{x}_i + \boldsymbol{\alpha}_c)}{B(\boldsymbol{\alpha}_c)} \tag{3.79}$$

(This equation is derived in Equation 5.24.) Surprisingly this simple change is all that is needed to capture the burstiness phenomenon. The intuitive reason for this is as follows: After seeing one occurence of a word, say word j, the posterior counts on θ_j gets updated, making another occurence of word j more likely. By contrast, if θ_j is fixed, then the occurences of each word are independent. The multinomial model corresponds to drawing a ball from an urn with K colors of ball, recording its color, and then replacing it. By contrast, the DCM model corresponds to drawing a ball, recording its color, and then replacing it with one additional copy; this is called the **Polya urn**.

Using the DCM as the class conditional density gives much better results than using the multinomial, and has performance comparable to state of the art methods, as described in (Madsen et al. 2005). The only disadvantage is that fitting the DCM model is more complex; see (Minka 2000e; Elkan 2006) for the details.

Exercises

Exercise 3.1 MLE for the Bernoulli/ binomial model

Derive Equation 3.22 by optimizing the log of the likelihood in Equation 3.11.

Exercise 3.2 Marginal likelihood for the Beta-Bernoulli model

In Equation 5.23, we showed that the marginal likelihood is the ratio of the normalizing constants:

$$p(D) = \frac{Z(\alpha_1 + N_1, \alpha_0 + N_0)}{Z(\alpha_1, \alpha_0)} = \frac{\Gamma(\alpha_1 + N_1)\Gamma(\alpha_0 + N_0)}{\Gamma(\alpha_1 + \alpha_0 + N)}\frac{\Gamma(\alpha_1 + \alpha_0)}{\Gamma(\alpha_1)\Gamma(\alpha_0)} \tag{3.80}$$

We will now derive an alternative derivation of this fact. By the chain rule of probability,

$$p(x_{1:N}) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})\ldots \tag{3.81}$$

In Section 3.3.4, we showed that the posterior predictive distribution is

$$p(X = k|D_{1:N}) = \frac{N_k + \alpha_k}{\sum_i N_i + \alpha_i} \triangleq \frac{N_k + \alpha_k}{N + \alpha} \tag{3.82}$$

where $k \in \{0, 1\}$ and $D_{1:N}$ is the data seen so far. Now suppose $D = H, T, T, H, H$ or $D = 1, 0, 0, 1, 1$. Then

$$p(D) = \frac{\alpha_1}{\alpha}\cdot\frac{\alpha_0}{\alpha+1}\cdot\frac{\alpha_0+1}{\alpha+2}\cdot\frac{\alpha_1+1}{\alpha+3}\cdot\frac{\alpha_1+2}{\alpha+4} \tag{3.83}$$

$$= \frac{[\alpha_1(\alpha_1+1)(\alpha_1+2)]\,[\alpha_0(\alpha_0+1)]}{\alpha(\alpha+1)\cdots(\alpha+4)} \tag{3.84}$$

$$= \frac{[(\alpha_1)\cdots(\alpha_1+N_1-1)]\,[(\alpha_0)\cdots(\alpha_0+N_0-1)]}{(\alpha)\cdots(\alpha+N-1)} \tag{3.85}$$

Show how this reduces to Equation 3.80 by using the fact that, for integers, $(\alpha - 1)! = \Gamma(\alpha)$.