

lecture6-naive-bayes

October 28, 2021

1 Lecture 6: Generative Models and Naive Bayes

1.0.1 Applied Machine Learning

Volodymyr Kuleshov Cornell Tech

2 Part 1: Text Classification

We will now do a quick detour to talk about an important application area of machine learning: text classification.

Afterwards, we will see how text classification motivates new classification algorithms.

3 Review: Classification

Consider a training dataset $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(n)}, y^{(n)})\}$.

We distinguish between two types of supervised learning problems depending on the targets $y^{(i)}$.

1. **Regression:** The target variable $y \in \mathcal{Y}$ is continuous: $\mathcal{Y} \subseteq \mathbb{R}$.
2. **Classification:** The target variable y is discrete and takes on one of K possible values: $\mathcal{Y} = \{y_1, y_2, \dots, y_K\}$. Each discrete value corresponds to a *class* that we want to predict.

4 Text Classification

An interesting instance of a classification problem is classifying text. * Includes a lot applied problems: spam filtering, fraud detection, medical record classification, etc. * Inputs x are sequences of words of an arbitrary length. * The dimensionality of text inputs is usually very large, proportional to the size of the vocabulary.

5 Classification Dataset: Twenty Newsgroups

To illustrate the text classification problem, we will use a popular dataset called **20-newsgroups**. * It contains ~20,000 documents collected approximately evenly from 20 different online newsgroups. * Each newsgroup covers a different topic such as medicine, computer graphics, or religion. * This dataset is often used to benchmark text classification and other types of algorithms.

Let's load this dataset.

```
[1]: #https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.
      ↪html

import numpy as np
import pandas as pd
from sklearn.datasets import fetch_20newsgroups

# for this lecture, we will restrict our attention to just 4 different
  ↪newsgroups:
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.
  ↪med']

# load the dataset
twenty_train = fetch_20newsgroups(subset='train', categories=categories,
  ↪shuffle=True, random_state=42)

# print some information on it
print(twenty_train.DESCR[:1100])
```

.. _20newsgroups_dataset:

The 20 newsgroups text dataset

The 20 newsgroups dataset comprises around 18000 newsgroups posts on 20 topics split in two subsets: one for training (or development) and the other one for testing (or for performance evaluation). The split between the train and test set is based upon a messages posted before and after a specific date.

This module contains two loaders. The first one, :func:`sklearn.datasets.fetch_20newsgroups`, returns a list of the raw texts that can be fed to text feature extractors such as :class:`sklearn.feature_extraction.text.CountVectorizer` with custom parameters so as to extract feature vectors. The second one, :func:`sklearn.datasets.fetch_20newsgroups_vectorized`, returns ready-to-use features, i.e., it is not necessary to use a feature extractor.

****Data Set Characteristics:****

=====	=====
Classes	20
Samples total	18846
Dimensionality	1
Features	text
=====	=====

Usage
~~~~~

```
[2]: # The set of targets in this dataset are the newgroup topics:
      twenty_train.target_names
```

```
[2]: ['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']
```

```
[3]: # Let's examine one data point
      print(twenty_train.data[3])
```

From: s0612596@let.rug.nl (M.M. Zwart)  
Subject: catholic church poland  
Organization: Faculteit der Letteren, Rijksuniversiteit Groningen, NL  
Lines: 10

Hello,

I'm writing a paper on the role of the catholic church in Poland after 1989.  
Can anyone tell me more about this, or fill me in on recent books/articles(  
in english, german or french). Most important for me is the role of the  
church concerning the abortion-law, religious education at schools,  
birth-control and the relation church-state(government). Thanx,

Masja,

"M.M.Zwart"<s0612596@let.rug.nl>

```
[4]: # We have about 2k data points in total
      print(len(twenty_train.data))
```

2257

## 6 Feature Representations for Text

Each data point  $x$  in this dataset is a squence of characters of an arbitrary length.

How do we transform these into  $d$ -dimensional features  $\phi(x)$  that can be used with our machine learning algorithms?

- We may devise hand-crafted features by inspecting the data:
  - Does the message contain the word “church”? Does the email of the user originate outside the United States? Is the organization a university? etc.
- We can count the number of occurrences of each word:
  - Does this message contain “Aardvark”, yes or no?

- Does this message contain “Apple”, yes or no?
- ... Does this message contain “Zebra”, yes or no?
- Finally, many modern deep learning methods can directly work with sequences of characters of an arbitrary length.

## 7 Bag of Words Representations

Perhaps the most widely used approach to representing text documents is called “bag of words”.

We start by defining a vocabulary  $V$  containing all the possible words we are interested in, e.g.:

$$V = \{\text{church, doctor, fervently, purple, slow, ...}\}$$

A bag of words representation of a document  $x$  is a function  $\phi(x) \rightarrow \{0, 1\}^{|V|}$  that outputs a feature vector

$$\phi(x) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ \vdots \end{pmatrix} \begin{matrix} \text{church} \\ \text{doctor} \\ \text{fervently} \\ \\ \text{purple} \\ \end{matrix}$$

of dimension  $V$ . The  $j$ -th component  $\phi(x)_j$  equals 1 if  $x$  contains the  $j$ -th word in  $V$  and 0 otherwise.

Let’s see an example of this approach on 20-newsgroups.

We start by computing these features using the `sklearn` library.

```
[5]: from sklearn.feature_extraction.text import CountVectorizer

# vectorize the training set
count_vect = CountVectorizer(binary=True)
X_train = count_vect.fit_transform(twenty_train.data)
X_train.shape
```

```
[5]: (2257, 35788)
```

In `sklearn`, we can retrieve the index of  $\phi(x)$  associated with each word using the expression `count_vect.vocabulary_.get(word)`:

```
[6]: # The CountVectorizer class records the index j associated with each word in V
print('Index for the word "church": ', count_vect.vocabulary_.get(u'church'))
print('Index for the word "computer": ', count_vect.vocabulary_.
      ↪get(u'computer'))
```

```
Index for the word "church": 8609
Index for the word "computer": 9338
```

Our featurized dataset is in the matrix `X_train`. We can use the above indices to retrieve the 0-1 value that has been computed for each word:

```
[7]: # We can examine if any of these words are present in our previous datapoint
print(twenty_train.data[3])

# let's see if it contains these two words?
print('---'*20)
print('Value at the index for the word "church": ', X_train[3, count_vect.
    ↪vocabulary_.get(u'church')])
print('Value at the index for the word "computer": ', X_train[3, count_vect.
    ↪vocabulary_.get(u'computer')])
print('Value at the index for the word "doctor": ', X_train[3, count_vect.
    ↪vocabulary_.get(u'doctor')])
print('Value at the index for the word "important": ', X_train[3, count_vect.
    ↪vocabulary_.get(u'important')])
```

From: s0612596@let.rug.nl (M.M. Zwart)  
Subject: catholic church poland  
Organization: Faculteit der Letteren, Rijksuniversiteit Groningen, NL  
Lines: 10

Hello,

I'm writing a paper on the role of the catholic church in Poland after 1989. Can anyone tell me more about this, or fill me in on recent books/articles( in english, german or french). Most important for me is the role of the church concerning the abortion-law, religious education at schools, birth-control and the relation church-state(government). Thanx,

Masja,

"M.M.Zwart"<s0612596@let.rug.nl>

```
-----
Value at the index for the word "church":  1
Value at the index for the word "computer": 0
Value at the index for the word "doctor":  0
Value at the index for the word "important": 1
```

## 8 Practical Considerations

In practice, we may use some additional modifications of this technique:

- Sometimes, the feature  $\phi(x)_j$  for the  $j$ -th word holds the count of occurrences of word  $j$  instead of just the binary occurrence.
- The raw text is usually preprocessed. One common technique is *stemming*, in which we only keep the root of the word.

- e.g. “slowly”, “slowness”, both map to “slow”
- Filtering for common *stopwords* such as “the”, “a”, “and”. Similarly, rare words are also typically excluded.

## 9 Classification Using BoW Features

Let’s now have a look at the performance of classification over bag of words features.

Now that we have a feature representation  $\phi(x)$ , we can apply the classifier of our choice, such as logistic regression.

```
[8]: from sklearn.linear_model import LogisticRegression

# Create an instance of Softmax and fit the data.
logreg = LogisticRegression(C=1e5, multi_class='multinomial', verbose=True)
logreg.fit(X_train, twenty_train.target)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 1.0s finished
```

```
[8]: LogisticRegression(C=100000.0, multi_class='multinomial', verbose=True)
```

And now we can use this model for predicting on new inputs.

```
[9]: docs_new = ['God is love', 'OpenGL on the GPU is fast']

X_new = count_vect.transform(docs_new)
predicted = logreg.predict(X_new)

for doc, category in zip(docs_new, predicted):
    print('%r => %s' % (doc, twenty_train.target_names[category]))
```

```
'God is love' => soc.religion.christian
'OpenGL on the GPU is fast' => comp.graphics
```

## 10 Summary of Text Classification

- Classifying text normally requires specifying features over the raw data.
- A widely used representation is “bag of words”, in which features are occurrences or counts of words.
- Once text is featurized, any off-the-shelf supervised learning algorithm can be applied, but some work better than others, as we will see next.

# Part 2: Generative Models

In this lecture, we are going to look at generative algorithms and their applications to text classification.

We will start by defining the concept of a generative *model*.

## 11 Review: Supervised Learning Models

A supervised learning model is a function

$$f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$$

that maps inputs  $x \in \mathcal{X}$  to targets  $y \in \mathcal{Y}$ .

Models have *parameters*  $\theta \in \Theta$  living in a set  $\Theta$ .

For example, logistic regression is a binary classification algorithm which uses a model

$$f_\theta : \mathcal{X} \rightarrow [0, 1]$$

of the form

$$f_\theta(x) = \sigma(\theta^\top x) = \frac{1}{1 + \exp(-\theta^\top x)},$$

where  $\sigma(z) = \frac{1}{1 + \exp(-z)}$  is the *sigmoid* or *logistic* function.

## 12 Review: Probabilistic Interpretations

Many supervised learning models have a probabilistic interpretation. Often a model  $f_\theta$  defines a probability distribution of the form

$$P_\theta(x, y) : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1] \quad \text{or} \quad P_\theta(y|x) : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1].$$

We refer to these as *probabilistic models*.

For example, our logistic model defines (“parameterizes”) a probability distribution  $P_\theta(y|x) : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$  as follows:

$$\begin{aligned} P_\theta(y = 1|x) &= \sigma(\theta^\top x) \\ P_\theta(y = 0|x) &= 1 - \sigma(\theta^\top x). \end{aligned}$$

## 13 Review: Conditional Maximum Likelihood

When a machine learning model defines a *conditional* model  $P_\theta(y|x)$ , we can maximize the *conditional maximum likelihood*:

$$\max_{\theta} \frac{1}{n} \sum_{i=1}^n \log P_\theta(y^{(i)}|x^{(i)}).$$

This says that we should choose parameters  $\theta$  such that for each input  $x^{(i)}$  in the dataset  $\mathcal{D}$ ,  $P_\theta$  assigns a high probability to the correct target  $y^{(i)}$ .

In the logistic regression example, we optimize the following objective defined over a binary classification dataset  $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(n)}, y^{(n)})\}$ .

$$\begin{aligned}\ell(\theta) &= \frac{1}{n} \sum_{i=1}^n \log P_{\theta}(y^{(i)} | x^{(i)}) \\ &= \frac{1}{n} \sum_{i=1}^n y^{(i)} \cdot \log \sigma(\theta^{\top} x^{(i)}) + (1 - y^{(i)}) \cdot \log(1 - \sigma(\theta^{\top} x^{(i)})).\end{aligned}$$

This objective is also often called the log-loss, or cross-entropy.

This asks the model to output a large score  $\sigma(\theta^{\top} x^{(i)})$  (a score that's close to one) if  $y^{(i)} = 1$ , and a score that's small (close to zero) if  $y^{(i)} = 0$ .

## 14 Discriminative Models

Logistic regression is an example of a *discriminative* machine learning model because \* It directly transforms  $x$  into a score for each class  $y$  (e.g., via the formula  $y = \sigma(\theta^{\top} x)$ ) \* It can be interpreted as defining a *conditional* probability  $P_{\theta}(y|x)$

## 15 Generative Models

Another approach to classification is to use *generative* models.

- A generative approach first builds a model of  $x$  for each class:

$$P_{\theta}(x|y = k) \text{ for each class } k.$$

$P_{\theta}(x|y = k)$  *scores* each  $x$  according to how well it matches class  $k$ .

- A class probability  $P_{\theta}(y = k)$  encoding our prior beliefs

$$P_{\theta}(y = k) \text{ for each class } k.$$

These are often just the % of each class in the data.

In the context of spam classification, we would fit two models on a corpus of emails  $x$  with spam/non-spam labels  $y$ :

$$P_{\theta}(x|y = 0) \qquad \text{and} \qquad P_{\theta}(x|y = 1)$$

as well as define priors  $P_{\theta}(y = 0), P_{\theta}(y = 1)$ .

$P_{\theta}(x|y = 1)$  *scores* each  $x$  based on how much it looks like spam.

$P_{\theta}(x|y = 0)$  *scores* each  $x$  based on how much it looks like non-spam.



## 16 Predictions From Generative Models

Given a new  $x'$ , we return the class that is the most likely to have generated it:

$$\begin{aligned}\arg \max_k P_\theta(y = k|x') &= \arg \max_k \frac{P_\theta(x'|y = k)P_\theta(y = k)}{P_\theta(x')} \\ &= \arg \max_k P_\theta(x'|y = k)P_\theta(y = k),\end{aligned}$$

where we have applied Bayes' rule in the first line.

In the context of spam classification, given a new  $x'$ , we would compare the probabilities of both models:

$$P_\theta(x'|y = 0)P_\theta(y = 0) \quad \text{vs.} \quad P_\theta(x'|y = 1)P_\theta(y = 1)$$

We output the class that's more likely to have generated  $x'$ .

## 17 Probabilistic Interpretations

A *generative* model defines  $P_\theta(x|y)$  and  $P_\theta(y)$ , thus it also defines a distribution of the form  $P_\theta(x, y)$ .

$$\underbrace{P_\theta(x, y) : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]}_{\text{generative model}} \qquad \underbrace{P_\theta(y|x) : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]}_{\text{discriminative model}}$$

Discriminative models don't define any probability over the  $x$ 's. Generative models do.

We can learn a generative model  $P_\theta(x, y)$  by maximizing the *likelihood*:

$$\max_{\theta} \frac{1}{n} \sum_{i=1}^n \log P_\theta(x^{(i)}, y^{(i)}).$$

This says that we should choose parameters  $\theta$  such that the model  $P_\theta$  assigns a high probability to each training example  $(x^{(i)}, y^{(i)})$  in the dataset  $\mathcal{D}$ .

## 18 Generative vs. Discriminative Approaches

What are the pros and cons of generative and discriminative methods?

- If we only care about prediction, we don't need a model of  $P(x)$ . It's simpler to only model  $P(y|x)$  (what we care about).
  - In practice, discriminative models are often be more accurate.
- If we care about other tasks (generation, dealing with missing values, etc.) or if we know the true model is generative, we want to use the generative approach.

More on this later!

## # Part 3: Naive Bayes

Next, we are going to look at Naive Bayes, a generative classification algorithm.

We will apply Naive Bayes to the text classification problem.

## 19 Review: Text Classification

An common type of classification problem is classifying text. \* Includes a lot applied problems: spam filtering, fraud detection, medical record classification, etc. \* Inputs  $x$  are sequences of words of an arbitrary length. \* The dimensionality of text inputs is usually very large, proportional to the size of the vocabulary.

## 20 A Generative Model for Text Classification

In binary text classification, we fit two models on a labeled corpus:

$$P_{\theta}(x|y=0) \quad \text{and} \quad P_{\theta}(x|y=1)$$

We also define priors  $P_{\theta}(y=0), P_{\theta}(y=1)$ .

Each model  $P_{\theta}(x|y=k)$  *scores*  $x$  based on how much it looks like class  $k$ . The documents  $x$  are in **bag-of-words** representation.

How do we choose  $P_{\theta}(x|y=k)$ ?

## 21 Review: Categorical Distribution

A [Categorical](#) distribution with parameters  $\theta$  is a probability over  $K$  discrete outcomes  $x \in \{1, 2, \dots, K\}$ :

$$P_{\theta}(x=j) = \theta_j.$$

When  $K=2$  this is called the [Bernoulli](#).

## 22 First Attempt at a Generative Model

Note there is a finite number of  $x$ 's: each is a binary vector of size  $d$ .

A first solution is to assume that  $P(x|y=k)$  is a categorical distribution that assigns a probability to each possible state of  $x$ :

$$P(x|y=k) = P_k \begin{pmatrix} 0 & \text{church} \\ 1 & \text{doctor} \\ 0 & \text{fervently} \\ \vdots & \vdots \\ 0 & \text{purple} \end{pmatrix} = \theta_{xk} = 0.0012$$

The  $\theta_{xk}$  is the probability of  $x$  under class  $k$ . We want to learn these.

## 23 Problem: High Dimensionality

How many parameters does a Categorical model  $P(x|y = k)$  have?

- If the dimensionality  $d$  of  $x$  is high (e.g., vocabulary has size 10,000),  $x$  can take a huge number of values ( $2^{10000}$  in our example)
- We need to specify  $2^d - 1$  parameters for the Categorical distribution.

For comparison, there are  $\approx 10^{82}$  atoms in the universe.

## 24 Naive Bayes: An Example

To deal with high-dimensional  $x$ , we choose a simpler model for  $P_\theta(y|x)$ : 1. We define a (Bernoulli) model with one parameter  $\psi_{jk} \in [0, 1]$  for the occurrence of each word  $j$  in class  $k$ :

$$P_\theta(x_j = 1 \mid y = k) = \psi_{jk}$$

$\psi_{jk}$  is the probability that a document of class  $k$  contains word  $j$ .

2. We define the model  $P_\theta(x|y = k)$  for documents  $x$  as the product of the occurrence probabilities of each of its words  $x_j$ :

$$P_\theta(x|y = k) = \prod_{j=1}^d P_\theta(x_j \mid y = k)$$

How many parameters does this new model have?

- We have a distribution  $P_\theta(x_j = 1 \mid y = k)$  for each word  $j$  and each distribution has one parameter  $\psi_{jk}$ .
- The distribution  $P_\theta(x|y = k) = \prod_{j=1}^d P_\theta(x_j \mid y = k)$  is the product of  $d$  such one-parameter distributions.
- We have  $K$  distributions of the form  $P_\theta(x|y = k)$ .

Thus, we only need  $Kd$  parameters instead of  $K(2^d - 1)!$

## 25 The Naive Bayes Assumption Machine Learning

The Naive Bayes assumption is a **general technique** that can be used with any  $d$ -dimensional  $x$ .

\* We simplify the model for  $x$  as:

$$P(x|y) = \prod_{j=1}^d P(x_j \mid y)$$

\* We choose a simple distribution family for  $P(x_j \mid y)$ .

This typically makes the number of parameters linear instead of exponential in  $d$ .

## 26 Is Naive Bayes a Good Assumption?

Naive Bayes assumes that words are uncorrelated, but in reality they are. \* If spam email contains “bank”, it probably contains “account”

As a result, the probabilities estimated by Naive Bayes can be over- under under-confident.

In practice, however, Naive Bayes is a very useful assumption that gives very good classification accuracy!

## 27 Defining Prior Distributions for Our Model

We still need to define the distribution  $P_\theta(y = k)$ . \* This encodes our prior belief about  $y$  before we see  $x$ . \* It can also be learned from data.

Since we have a small number of classes  $K$ , we may use a Categorical distribution with parameters  $\vec{\phi} = (\phi_1, \dots, \phi_K)$  and learn  $\vec{\phi}$  from data:

$$P_\theta(y = k) = \phi_k.$$

## 28 Bernoulli Naive Bayes Model

The *Bernoulli Naive Bayes* model  $P_\theta(x, y)$  is defined for *binary data*  $x \in \{0, 1\}^d$  (e.g., bag-of-words documents).

The  $\theta$  contains prior parameters  $\vec{\phi} = (\phi_1, \dots, \phi_K)$  and  $K$  sets of per-class parameters  $\psi_k = (\psi_{1k}, \dots, \psi_{dk})$ .

The probability of the data  $x$  for each class equals

$$P_\theta(x|y = k) = \prod_{j=1}^d P(x_j | y = k),$$

where each  $P_\theta(x_j | y = k)$  is a Bernoulli( $\psi_{jk}$ ).

The probability over  $y$  is Categorical:  $P_\theta(y = k) = \phi_k$ .

Formally, we have:

$$\begin{aligned} P_\theta(y) &= \text{Categorical}(\phi_1, \phi_2, \dots, \phi_K) \\ P_\theta(x_j = 1|y = k) &= \text{Bernoulli}(\psi_{jk}) \\ P_\theta(x|y = k) &= \prod_{j=1}^d P_\theta(x_j|y = k) \end{aligned}$$

The parameters of the model are  $\theta = (\phi_1, \dots, \phi_K, \psi_{11}, \dots, \psi_{dK})$ . There are exactly  $K(d + 1)$  parameters.

# Part 4: Naive Bayes: Learning

We will now turn our attention to learning the parameters of the Naive Bayes model and using them to make predictions.

## 29 Review: Maximum Likelihood Learning

We can learn a generative model  $P_\theta(x, y)$  by maximizing the *maximum likelihood*:

$$\max_{\theta} \frac{1}{n} \sum_{i=1}^n \log P_\theta(x^{(i)}, y^{(i)}).$$

This seeks to find parameters  $\theta$  such that the model assigns high probability to the training data.

Let's use maximum likelihood to fit the Bernoulli Naive Bayes model. Note that model parameters  $\theta$  are the union of the parameters of each sub-model:

$$\theta = (\phi_1, \phi_2, \dots, \phi_K, \psi_{11}, \psi_{21}, \dots, \psi_{dK}).$$

## 30 Learning a Bernoulli Naive Bayes Model

Given a dataset  $\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid i = 1, 2, \dots, n\}$ , we want to optimize the log-likelihood  $\ell(\theta) = \log L(\theta)$ :

$$\begin{aligned} \ell &= \sum_{i=1}^n \log P_\theta(x^{(i)}, y^{(i)}) = \sum_{i=1}^n \sum_{j=1}^d \log P_\theta(x_j^{(i)} | y^{(i)}) + \sum_{i=1}^n \log P_\theta(y^{(i)}) \\ &= \sum_{k=1}^K \sum_{j=1}^d \underbrace{\sum_{i: y^{(i)}=k} \log P(x_j^{(i)} | y^{(i)}; \psi_{jk})}_{\text{all the terms that involve } \psi_{jk}} + \underbrace{\sum_{i=1}^n \log P(y^{(i)}; \vec{\phi})}_{\text{all the terms that involve } \vec{\phi}}. \end{aligned}$$

In equality #2, we use Naive Bayes:  $P_\theta(x, y) = P_\theta(y) \prod_{i=1}^d P(x_i | y)$ ; in the third one, we change the order of summation.

Each  $\psi_{jk}$  for  $k = 1, 2, \dots, K$  is found in only the following terms:

$$\max_{\psi_{jk}} \ell(\theta) = \max_{\psi_{jk}} \sum_{i: y^{(i)}=k} \log P(x_j^{(i)} | y^{(i)}; \psi_{jk}).$$

Thus, optimization over  $\psi_{jk}$  can be carried out independently of all the other parameters by just looking at these terms.

Similarly, optimizing for  $\vec{\phi} = (\phi_1, \phi_2, \dots, \phi_K)$  only involves a few terms:

$$\max_{\vec{\phi}} \sum_{i=1}^n \log P_\theta(x^{(i)}, y^{(i)}; \theta) = \max_{\vec{\phi}} \sum_{i=1}^n \log P_\theta(y^{(i)}; \vec{\phi}).$$

## 31 Learning the Parameters $\phi$

Let's first consider the optimization over  $\vec{\phi} = (\phi_1, \phi_2, \dots, \phi_K)$ .

$$\max_{\vec{\phi}} \sum_{i=1}^n \log P_\theta(y = y^{(i)}; \vec{\phi}).$$

\* We have  $n$  datapoints, each having one of  $K$  classes \* We want to learn the most likely class probabilities  $\phi_k$  that generated this data

What is the maximum likelihood  $\phi$  in this case?

Intuitively, the maximum likelihood class probabilities  $\phi$  should just be the class proportions that we see in the data.

Let's calculate this formally. Our objective  $J(\vec{\phi})$  equals

$$\begin{aligned} J(\vec{\phi}) &= \sum_{i=1}^n \log P_{\theta}(y^{(i)}; \vec{\phi}) = \sum_{i=1}^n \log \left( \frac{\phi_{y^{(i)}}}{\sum_{k=1}^K \phi_k} \right) \\ &= \sum_{i=1}^n \log \phi_{y^{(i)}} - n \cdot \log \sum_{k=1}^K \phi_k \\ &= \sum_{k=1}^K \sum_{i: y^{(i)}=k} \log \phi_k - n \cdot \log \sum_{k=1}^K \phi_k \end{aligned}$$

Taking the derivative and setting it to zero, we obtain

$$\frac{\phi_k}{\sum_l \phi_l} = \frac{n_k}{n}$$

for each  $k$ , where  $n_k = |\{i : y^{(i)} = k\}|$  is the number of training targets with class  $k$ .

Thus, the optimal  $\phi_k$  is just the proportion of data points with class  $k$  in the training set!

## 32 Learning the Parameters $\psi_{jk}$

Next, let's look at the maximum likelihood term

$$\arg \max_{\psi_{jk}} \sum_{i: y^{(i)}=k} \log P(x_j^{(i)} | y^{(i)}; \psi_{jk}).$$

over the word parameters  $\psi_{jk}$ .

- Our dataset are all the inputs  $x$  for which  $y = k$ .
- We seek the probability  $\psi_{jk}$  of a word  $j$  being present in a  $x$ .

What is the maximum likelihood  $\psi_{jk}$  in this case?

Each  $\psi_{jk}$  is simply the proportion of documents in class  $k$  that contain the word  $j$ .

We can maximize the likelihood exactly like we did for  $\phi$  to obtain closed form solutions:

$$\psi_{jk} = \frac{n_{jk}}{n_k}.$$

where  $|\{i : x_j^{(i)} = 1 \text{ and } y^{(i)} = k\}|$  is the number of  $x^{(i)}$  with label  $k$  and a positive occurrence of word  $j$ .

## 33 Querying the Model

How do we ask the model for predictions? As discussed earlier, we can apply Bayes' rule:

$$\arg \max_y P_\theta(y|x) = \arg \max_y P_\theta(x|y)P(y).$$

Thus, we can estimate the probability of  $x$  and under each  $P_\theta(x|y = k)P(y = k)$  and choose the class that explains the data best.

## 34 Classification Dataset: Twenty Newsgroups

To illustrate the text classification problem, we will use a popular dataset called **20-newsgroups**. \* It contains ~20,000 documents collected approximately evenly from 20 different online newsgroups. \* Each newsgroup covers a different topic such as medicine, computer graphics, or religion. \* This dataset is widely used to benchmark text classification and other types of algorithms.

Let's load this dataset.

```
[10]: #https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.
      ↪html

import numpy as np
import pandas as pd
from sklearn.datasets import fetch_20newsgroups

# for this lecture, we will restrict our attention to just 4 different
↪newsgroups:
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.
↪med']

# load the dataset
twenty_train = fetch_20newsgroups(subset='train', categories=categories,
↪shuffle=True, random_state=42)

# print some information on it
print(twenty_train.DESCR[:1100])
```

```
.. _20newsgroups_dataset:
```

The 20 newsgroups text dataset

-----

The 20 newsgroups dataset comprises around 18000 newsgroups posts on 20 topics split in two subsets: one for training (or development) and the other one for testing (or for performance evaluation). The split between the train and test set is based upon a messages posted before and after a specific date.

This module contains two loaders. The first one, `:func:`sklearn.datasets.fetch_20newsgroups``, returns a list of the raw texts that can be fed to text feature extractors such as `:class:`sklearn.feature_extraction.text.CountVectorizer`` with custom parameters so as to extract feature vectors. The second one, `:func:`sklearn.datasets.fetch_20newsgroups_vectorized``, returns ready-to-use features, i.e., it is not necessary to use a feature extractor.

**\*\*Data Set Characteristics:\*\***

|                |       |
|----------------|-------|
| Classes        | 20    |
| Samples total  | 18846 |
| Dimensionality | 1     |
| Features       | text  |

Usage

~~~~~

35 Example: Text Classification

Let's see how this approach can be used in practice on the text classification dataset. * We will learn a good set of parameters for a Bernoulli Naive Bayes model * We will compare the outputs to the true predictions.

Let's see an example of Naive Bayes on 20-newsgroups.

We start by computing these features using the `sklearn` library.

```
[11]: from sklearn.feature_extraction.text import CountVectorizer

# vectorize the training set
count_vect = CountVectorizer(binary=True, max_features=1000)
y_train = twenty_train.target
X_train = count_vect.fit_transform(twenty_train.data).toarray()
X_train.shape
```

```
[11]: (2257, 1000)
```

Let's compute the maximum likelihood model parameters on our dataset.

```
[12]: # we can implement these formulas over the Iris dataset
n = X_train.shape[0] # size of the dataset
d = X_train.shape[1] # number of features in our dataset
K = 4 # number of classes
```



```

# these are the shapes of the parameters
psis = np.zeros([K,d])
phis = np.zeros([K])

# we now compute the parameters
for k in range(K):
    X_k = X_train[y_train == k]
    psis[k] = np.mean(X_k, axis=0)
    phis[k] = X_k.shape[0] / float(n)

# print out the class proportions
print(phis)

```

```
[0.21267169 0.25875055 0.26318121 0.26539654]
```

We can compute predictions using Bayes' rule.

```

[13]: # we can implement this in numpy
def nb_predictions(x, psis, phis):
    """This returns class assignments and scores under the NB model.

    We compute  $\arg\max_y p(y/x)$  as  $\arg\max_y p(x/y)p(y)$ 
    """
    # adjust shapes
    n, d = x.shape
    x = np.reshape(x, (1, n, d))
    psis = np.reshape(psis, (K, 1, d))

    # clip probabilities to avoid log(0)
    psis = psis.clip(1e-14, 1-1e-14)

    # compute log-probabilities
    logpy = np.log(phis).reshape([K,1])
    logpxy = x * np.log(psis) + (1-x) * np.log(1-psis)
    logpyx = logpxy.sum(axis=2) + logpy

    return logpyx.argmax(axis=0).flatten(), logpyx.reshape([K,n])

idx, logpyx = nb_predictions(X_train, psis, phis)
print(idx[:10])

```

```
[1 1 3 0 3 3 3 2 2 2]
```

We can measure the accuracy:

```
[14]: (idx==y_train).mean()
```

```
[14]: 0.8692955250332299
```

```
[15]: docs_new = ['OpenGL on the GPU is fast']

X_new = count_vect.transform(docs_new).toarray()
predicted, logpyx_new = nb_predictions(X_new, psis, phis)

for doc, category in zip(docs_new, predicted):
    print('%r => %s' % (doc, twenty_train.target_names[category]))
```

'OpenGL on the GPU is fast' => comp.graphics

36 Algorithm: Bernoulli Naive Bayes

- **Type:** Supervised learning (multi-class classification)
- **Model family:** Products of Bernoulli distributions, categorical priors
- **Objective function:** Log-likelihood.
- **Optimizer:** Closed form solution.