Linear Text classification

Problem defintion: Given a text document, assign it a discrete label $y \in \mathcal{Y}$ where $\mathcal{Y}$ is the set of possible labels. Many possible applications:

- Spam filter: $\mathcal{Y} = \{\text{Spam, non-spam}\}$
- Sentiment: $\mathcal{Y} = \{\text{Positive, negative, neutral}\}$
- Genre classification: $\mathcal{Y} = \{\text{sports, fiction, news}, \cdots\}$

Bag-of-words representation of a document

A typical representation of a document is a bag of words, which is mathematically a vector of word counts:

$$\mathbf{x} = [1, 5, 0, 3, 19, 0, \ldots]$$

where x_j is the count of the word j. The length of the vector is the size of the vocabulary $|V|$. (So that the vector for all documents in a set are of the same length for apple-to-apple comparison.)

- "Vocabulary" here can't be understood as words in a language. For instance, it could include all bigrams in a collection of documents.
- Alternatives to word counts include simple presence 1 or absence 0 of a word, tf/idf of a word, etc.
- By using word count we dropped all word order information

Feature function

- Not all words are equally important for purposes of predicting a particular label. To predict the label of a document, we assign a score to each word in the vocabulary to indicate the "compatibility" with the label, e.g., "basketball" has a high compatibility with *sports*, "Gryffindor" has a high compatibility with *fiction*.
- These compatibility scores are called *weights* and they are arranged in a vector $\boldsymbol{\theta}$.
- Given a bag-of-words $\boldsymbol{x}$ and a weight vector $\boldsymbol{\theta}$, we predict the label y by computing the total compatibility score between $\boldsymbol{x}$ and y.
- In a linear function, this compatibility score is the inner product of between the weights $\boldsymbol{\theta}$ and a *feature function*:

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} \boldsymbol{\Psi}(\boldsymbol{x}, y)$$

$$\boldsymbol{\Psi}(\boldsymbol{x}, y) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y) = \sum_{i} \theta_j f_j(\boldsymbol{x}, y)$$

More on feature function

- The feature function has two arguments, the word counts $\boldsymbol{x}$ and the label y.
- It will return an feature vector where each element of the vector might be:

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

$$f_j(\boldsymbol{x}, y) = \begin{cases} x_{whale}, & \text{if} \quad y = \text{Fiction} \\ 0, & \text{Otherwise} \end{cases}$$

- In this case the size of the feature vector is the size of the vocabulary, but it doesn't have to be.
- The output of the feature function also doesn't have to be the word count.

Shape of the feature vector

$$\boldsymbol{f}(\boldsymbol{x}, y=1) = [\boldsymbol{x}; \underbrace{0; 0; \cdots ; 0}_{(K-1)\times V}]$$

$$\boldsymbol{f}(\boldsymbol{x}, y=2) = [\underbrace{0; 0; \cdots ; 0}_{V}; \boldsymbol{x}; \underbrace{0; 0; \cdots ; 0}_{(K-2)\times V}]$$

$$\boldsymbol{f}(\boldsymbol{x}, y=K) = [\underbrace{0; 0; \cdots ; 0}_{(K-1)\times V}; \boldsymbol{x}]$$

https://powcoder.com
Add WeChat powcoder

where K is size of the label set, $\underbrace{0; 0; \cdots ; 0}_{(K-1)\times V}$ is a vector of $(K-1) \times V$ zeros and the semicolon indicates vertical concatenation.

Note: Think of a feature vector this way is good for mathematical presentation. It doesn't have to be implemented this way.

Bias

It is common to add a *offset feature* or *bias* at the end of the vector of word counts which is always 1, and then we need to add a zero to each of the zero vectors, to make the vector lengths match. The entire vector $\boldsymbol{f}(\boldsymbol{x}, y)$ will be $(V+1)K$

Assignment Project Exam Help

$$\boldsymbol{f}(\boldsymbol{x}, y=1) = [\boldsymbol{x};\ \underbrace{0; 0; \cdots; 0}_{(K-1)\times(V+1)}\ ]$$

https://powcoder.com

$$\boldsymbol{f}(\boldsymbol{x}, y=2) = [\underbrace{0; 0; \cdots; 0}_{V}\ \boldsymbol{x};\ \underbrace{0; 0; \cdots; 0}_{(K-2)\times(V+1)}\ ]$$

Add WeChat powcoder

$$\boldsymbol{f}(\boldsymbol{x}, y=K) = [\ \underbrace{0; 0; \cdots; 0}_{(K-1)\times(V+1)}\ ; \boldsymbol{x}]$$

Bias: What's its effect on classification if it is the only feature?

Example "vocabulary" V

Vocabulary for a collection of documents

Assignment Project Exam Help

A	NEG	not funny at all
B	NEG	painful, not funny
C	NEU	ok overall
D	POS	funny story
E	POS	good story, good jokes

https://powcoder.com

Add WeChat powcoder

$V = \{$ not, funny, painful, ok, overall, story, good, jokes $\}$

From vocabulary to feature function

Vocabulary for a collection of documents

A	NEG	not funny at all
B	NEG	painful, not funny
C	NEU	ok overall
D	POS	funny story
E	POS	good story, good jokes

$V = \{$ not, funny, painful, ok, overall, story, good, jokes $\}$

$$f_1(\boldsymbol{x}, y) = \begin{cases} x_{not}, & \text{if} \quad y = \text{NEG} \\ 0, & \text{Otherwise} \end{cases}$$

From vocabulary to feature function

Vocabulary for a collection of documents

A	NEG	not funny at all
B	NEG	painful, not funny
C	NEU	ok overall
D	POS	funny story
E	POS	good story, good jokes

$V = \{$ not, funny, painful, ok, overall, story, good, jokes $\}$

$$f_2(\boldsymbol{x}, y) = \begin{cases} x_{funny}, & \text{if} \quad y = \text{NEG} \\ 0, & \text{Otherwise} \end{cases}$$

From vocabulary to feature function

Vocabulary for a collection of documents

A	NEG	not funny at all
B	NEG	painful, not funny
C	NEU	ok overall
D	POS	funny story
E	POS	good story, good jokes

$V = \{$ not, funny, painful, ok, overall, story, good, jokes $\}$

Feature vector for A:

$$\boldsymbol{f}(\boldsymbol{x} = \text{featurize}(A), y = NEG) = [1; 1; 0; 0; 0; 0; 0; 0; 1; \underbrace{0; 0; \cdots ; 0}_{(3-1)\times(8+1)}]$$

From vocabulary to feature function

Vocabulary for a collection of documents

A	NEG	not funny at all
B	NEG	painful, not funny
C	NEU	ok overall
D	POS	funny story
E	POS	good story, good jokes

$V = \{$ not, funny, painful, ok, overall, story, good, jokes $\}$
Feature vector for C:

$$\boldsymbol{f}(\boldsymbol{x} = \text{featurize}(C), y = NEU) =$$

$$[\underbrace{0; 0; \cdots; 0}_{(8+1)}; 0; 0; 0; 1; 1; 0; 0; 0; 1; \underbrace{0; 0; \cdots; 0}_{(8+1)}]$$

From vocabulary to feature function

Vocabulary for a collection of documents

A	NEG	not funny at all
B	NEG	painful, not funny
C	NEU	ok overall
D	POS	funny story
E	POS	good story, good jokes

Assignment Project Exam Help
https://powcoder.com

$V = \{$ not, funny, painful, ok, overall, story, good, jokes $\}$
Feature vector for E

Add WeChat powcoder

$$\boldsymbol{f}(\boldsymbol{x} = \text{featurize}(E), y = POS) =$$
$$[\underbrace{0; 0; \cdots; 0}_{(3-1)\times(8+1)}; 0; 0; 0; 0; 0; 0; 1; 1; 1]$$

The importance of feature functions

- The performance of a model to a large extent depends on the use of proper feature functions

- A lot of research went to find the most effective features when developing machine learning systems

- Models that can't handle a large number of features usually don't perform as well

- The most effective features differ from task to task, and hence relies on a good understanding of the problem at hand and domain knowledge

Assigning weights to features

Now we know about features. What about the weights ($\boldsymbol{\theta}$)?

$$\Psi(\boldsymbol{x}, y) = \boldsymbol{f}(\boldsymbol{x}, y)\boldsymbol{\theta}$$

- There are many different ways to estimate the weights $\boldsymbol{\theta}$. That's why we have different machine learning models.

- We find the optimal value of $\boldsymbol{\theta}$ with a set of training samples of size N: $\{\boldsymbol{x}^{1:N}, y^{1:N}\}$

Probability preliminaries

- Joint probability: $P(X = a, Y = b)$ or $P(a, b)$ where X and Y are random variables and a and b are values assigned to the random variables
- Conditional probability: $P(X = a|Y = b) = \frac{P(X=a, Y=b)}{P(Y=b)}$
- Bayes' theorem: $P(X = a|Y = b) = \frac{P(Y=b|X=a)P(X=a)}{P(Y=b)}$
- Marginalization (sum rule): $P(Y) = \sum_X P(X, Y)$
- Independence: $P(Y|X) = P(Y)$, $P(X|Y) = P(X)$, $P(X, Y) = P(X)P(Y)$
- Conditional independence: $P(X, Y|Z) = P(X|Z)P(Y|Z)$

Naïve Bayes: the objective

The objective is to maximize the joint probability of a set of labeled training documents $p(\boldsymbol{x}^{1:N}, y^{1:N})$, where N is the number of documents. This is known as the **maximum likelihood estimation**.

The goal of the training process is to find the weights $\boldsymbol{\theta}$ that maximizes this likelihood:

$$
\begin{aligned}
\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{x}^{1:N}, y^{1:N}; \boldsymbol{\theta}) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{N} p(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{N} \log p(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})
\end{aligned}
$$

The notation $p(\boldsymbol{x}^{1:N}, y^{1:N}; \boldsymbol{\theta})$ indicate that $\boldsymbol{\theta}$ is a parameter of the probability function. Symbols in bold indicate a vector of variables rather than a single variable.

“Independent and Identically Distributed”

- “A collection of random variables is **independent and identically distributed** if each random variable has the same probability distribution as the others and all are mutually independent.” -from Wikipedia
- Often shortened as *i.i.d*
- The basis on which we can break down the joint probability of all samples into the product of the probability of each sample

The generative story of Naïve Bayes

The probability $p(\boldsymbol{x}^{1:N}, y^{1:N}; \boldsymbol{\theta})$ is defined through a **generative model**, an idealized random process that has generated the observed data. The algorithm that describes the generative model underlying the Naïve Bayes classifier with the parameters $\Theta = \{\boldsymbol{\mu}, \boldsymbol{\phi}\}$:

Algorithm 1 Generative process for the Naïve Bayes classification model

for Instance $i \in \{1, 2, \cdots, N\}$ **do**
  Draw the label $y^{(i)} \propto \text{Categorical}(\boldsymbol{\mu})$;
  Draw the word counts $\boldsymbol{x}^{(i)} | y^{(i)} \propto \text{Multinomial}(\boldsymbol{\phi}^{(i)})$.
end for

Multinomial distribution

$$P_{X,Y}(\boldsymbol{x}^{(i)}, y^{(i)}) = P_{X|Y}(\boldsymbol{x}^{(i)} \mid y^{(i)}) \times P_Y(y^{(i)})$$

$P_{Y|X}$ is a **multinomial**, which a probabilistic distribution over vectors of non-negative counts. The probability mass function for this distribution is:

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

$$P_{multi}(\boldsymbol{x}, \phi) = B(\boldsymbol{x}) \prod_{j=1}^{V} \phi_j^{x_j}$$

$$B(\boldsymbol{x}) = \frac{\left(\sum_{j=1}^{V} x_j\right)!}{\prod_{j=1}^{V}(x_j!)}$$

Crucially, $B(\boldsymbol{x})$ is a multinomial coefficient that does not depend on ϕ, and can usually be ignored.

Parameter estimation

The generative story above allows us to decompose

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

into

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\mu}) = \sum_{i=1}^{N} \log P_{mult}(\boldsymbol{x}^{(i)}; \boldsymbol{\phi}_{y^{(i)}}) + \log P_{cat}(y^{(i)}; \boldsymbol{\mu})$$

$$= \sum_{i=1}^{N} \log B(\boldsymbol{x}^{(i)}) + \sum_{j=1}^{V} x_j^{(i)} \log \phi_{y^{(i)},j} + \sum_{i=1}^{N} \log \mu_{y^{(i)}}$$

Maximum-likelihood estimation chooses $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$ that maximize the log-likelihood of $\mathcal{L}$. Because we want these parameters to be probabilities, the solution must obey the following constraints:

$$\sum_{j=1}^{V} \phi_{y,j} = 1 \quad \forall y$$

Parameter Estimation

After incorporating the the constraints by adding a set of Lagrange multipliers, we get this new objective (we focus on the $\phi_{y,j}$ for now):

$$\ell(\Phi_y) = \sum_{i:y^{(i)}=y} \sum_{j=1}^{V} x_j^{(i)} \log \phi_{y,j} - \lambda \left(\sum_{j=1}^{V} \phi_{y,j} - 1 \right)$$

Differentiating with respect to the parameters $\phi_{y,j}$ yields,

$$\frac{\partial \ell(\phi_y)}{\partial \phi}_{y,j} = \sum_{i:y^{(i)}=y} x_j^{(i)} / \phi_{y,j} - \lambda$$

Parameter Estimation

There is a closed form solution to this, which can be obtained by setting each element in the vector of derivatives equal to zero:

$$\lambda \phi_{y,j} = \sum_{i:y^{(i)}=y} x_j^{(i)}$$

$$\phi_{y,j} \propto \sum_{i:y^{(i)}=y} x_j^{(i)} = \sum_{i=1}^{N} \delta\left(y^{(i)} = y\right) x_j^{(i)} = count(y,j)$$

where $\delta\left(y^{(i)} = y\right)$ is an indicator function which returns one if $y^{(i)} = y$. The symbol $\propto$ indicates that $\phi_{y,j}$ is proportional to the right-hand side of the equation

Parameter Estimation

- Recall the constraint that ϕ_y is a vector of probabilities: $\sum_{j=1}^{V} \phi_{y,j} = 1$. We have an exact solution:

$$\phi_{y,j} = \frac{count(y,j)}{\sum_{j'=1}^{V} count(y,j')}$$

- Similarly we can arrive at:

$$\mu_y = \frac{count(y)}{\sum_{y' \in K} count(y')}$$

As is often the case, the result of the mathematical derivation (that needs to turned into code) is usually often much simpler:

Smoothing

Laplace smoothing: add α

$$\phi_{y,j} = \frac{\alpha + count(y,j)}{V\alpha + \sum_{j'=1}^{V} count(y,j')}$$

The **bias - variance** trade-off:

- Unbiased classifiers may **overfit** the training data, yielding poor performance on unseen data.

- But if the smoothing is too large, the resulting classifier can **underfit** instead. In the limit of $\alpha \to \infty$, there is zero variance: you get the same classifier, regardless of the data.

How to determine the best α?

Grid Search

Setting hyperparameters with grid search: parameters and hyperparameters

- Try a set of values and find the one that maximizes the accuracy, but on which data set?
- The goal is to maximize the system on *unseen* data, so we shouldn't be doing it on training data. Instead, we should do it on a *development* or *tuning* set.
- We should also set aside another set called *test* set that you measure system performance on.
- If the data set is too small, use *cross-validation*

Prediction with Naïve Bayes

$$
\begin{aligned}
\hat{y} &= \operatorname{argmax} \log \boldsymbol{p}(\boldsymbol{x}, y; \boldsymbol{\mu}, \phi) \\
&= \operatorname{argmax} \log p(\boldsymbol{x} | y; \phi) + \log \boldsymbol{p}(y; \boldsymbol{\mu})
\end{aligned}
$$

Plug in the distribution from the generative story, we get:

$$
\begin{aligned}
\log p(\boldsymbol{x} | y; \phi) + \log \boldsymbol{p}(y; \boldsymbol{\mu}) &= \log \left[B(\boldsymbol{x}) \prod_{j=1}^{V} \phi_{y,j}^{x_j} \right] + \log \mu_y \\
&= \log B(\boldsymbol{x}) + \sum_{j=1}^{V} x_j \log(\phi_{y,j}) + \log \mu_y \\
&= \log B(\boldsymbol{x}) + \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y),
\end{aligned}
$$

where

$$
\begin{aligned}
\boldsymbol{\theta} &= [\boldsymbol{\theta}^{(1)}; \boldsymbol{\theta}^{(2)}; \cdots; \boldsymbol{\theta}^{(K)}] \\
\boldsymbol{\theta}^{(y)} &= [\log \phi_{y,1}; \log \phi_{y,2}; \cdots; \log \phi_{y,V}; \log \mu_y]
\end{aligned}
$$

Relation between mathematical models and computer science tools

- Mathematics provides the justification of why a model works the way it does, and computer science focuses on realizing it with efficient algorithms and appropriate data structures.
  - Computational algorithms often resort to caching to avoid repeated computation, thus making the computational implementation more efficient, e.g., Viterbi, Forward-Backfoward, CKY

- It's useful to think about where the mathematical justification ends and computational realization starts.
- The relation between a mathematical expression and its implementation in a programming language can be thought of as a translation process: it's not always word for word.

Advantages of Naïve Bayes

With a joint likelihood objective, the estimated parameters of a Naïve Bayes model can be used for both classification ($p(y|x)$) and generation ($p(x|y)$).

$$P(x, y) = P(x|y) \times P(y) = P(y|x) \times P(x)$$

In practice, we rarely use Naïve Bayes for generation. It's mostly used as a classification model.

Problems of Naïve Bayes

Let's say we want to include some subword units (e.g., morphemes).

$$
\begin{aligned}
&P(\text{word} = \textit{unfit}, \text{prefix} = \textit{un-}|y) \\
&= P(\text{prefix} = \textit{un-}|\text{word} = \textit{unfit}, y) \times P(\text{word} = \textit{unfit}|y) \\
&= 1 \times P(\text{word} = \textit{unfit}|y)
\end{aligned}
$$

If we assume conditional independence,

$$
\begin{aligned}
&P(\text{word} = \textit{unfit}, \text{prefix} = \textit{un-}|y) \\
&\approx P(\text{word} = \textit{unfit}|y) \times P(\text{prefix} = \textit{un-}|y)
\end{aligned}
$$

Since $P(\text{word} = \textit{unfit}|y) \geq P(\text{word} = \textit{unfit}|y) \times P(\text{prefix} = \textit{un-}|y)$, conditional independence under-estimates the true probabilities of conjunction of positively correlated features.