Classification with Naive Bayes

Benjamin Roth
(Many thanks to Helmut Schmid for parts of the slides)

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilian-Universität München
beroth@cis.uni-muenchen.de

Mathematical basics

Random experiment

Statistics are about the probability of events:

Example: How likely is it to have six rightists in the lottery?

Random experiment: Experiment (experiment) with several possible outputs (*throw of two cubes*)

Result: Result of an experiment (*3 eyes on dice 1 and 4 eyes on dice 2*)

Sample space Ω: Set of all possible results

Event: Subset of the sample space (*7 eyes on two dice*)

Sample: Series of results in a repeated experiment

Probability distribution

Probability distribution: Function that assigns each result a value between 0 and 1, so that

$$\sum_{o \in \Omega} p(o) = 1$$

The probability of a event is the sum of the probabilities of the corresponding results.

Example:

probability that the number of eyes when throwing a dice is straight

Conditional and a priori probability

Conditional probability: Probability of an event A, if the event B is known:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Example: Probability that the number of points in a dice is even if the number of points is greater than 3

A priori probability $P(A)$**:** Probability of event A without knowing event B

Random variables

Random variable: Function, which assigns each result a real number.

Example: Mapping of grades *very good, good, satisfactory, sufficient, poor, insufficient* to the numbers 1, 2, 3, 4, 5, 6

A random variable is called discrete if it takes only finitely many or countably infinite values.

The above example thus describes a discrete random variable.

Probability of a value x of the random variable X:

$$P(X = x) = p(x) = P(A_x)$$

A random variable with only the values 0 and 1 is called **Bernoulli experiment**.

Common distributions and marginal distributions

The **common distribution** of two random variables X and Y:

$$p(x,y) = P(X = x, Y = y) = P(A_x \cap A_y)$$

The **marginal distribution** of two random variables X and Y:

$$p_X(x) = \sum_y p(x,y) \quad p_Y(y) = \sum_x p(x,y)$$

Independence: The random variables X and Y are statistically independent if:

$$p(x,y) = p_X(x)p_Y(y)$$

Example: When throwing two dice, their numbers are statistically independent of each other.

Important rules

Chain rule: Some common probabilities can be converted into a product of conditional probabilities.

$$\begin{aligned} P(A_1 \cap A_2 \cap ... \cap A_n) &= P(A_1)P(A_2|A_1)...P(A_n|A_1 \cap ... \cap A_{n-1}) \\ &= \prod_{i=1}^{n} P(A_i|A_1 \cap ... \cap A_{i-1}) \end{aligned}$$

Theorem of Bayes: allows to "reverse" a conditional probability

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Probability estimation

$$\tilde{p}(x) = \frac{n(x)}{N}$$

The **relative frequency** $n(x)/N$ is the number of occurrences (*counts*) $n(x)$ an event x divided by the sample size n.

For increasing sample size n, the relative frequency converges to the actual probability of an event.

More precisely: the probability that the relative frequency differs more than ϵ from the actual probability converges to 0 for increasing sample size.

Probability estimation by relative frequency

Example:

- Random event: word occurrence is a specific word
- $n(x)$: Number of occurrences (*counts*) of the word in a corpus
- N: Number of word occurrences in the corpus.

word	n(word)	$\tilde{p}$(word)
meet		
deadline		
single		
...		

Probability estimation by relative frequency

Example:

- Random event: word occurrence is a specific word
- $n(x)$: Number of occurrences (*counts*) of the word in a corpus
- N: Number of word occurrences in the corpus.

word	n(word)	$\tilde{p}$(word)
meet	2	$\frac{2}{15} \approx 0.133$
deadline	2	$\frac{2}{15} \approx 0.133$
single	1	$\frac{1}{15} \approx 0.067$
...		

Relative frequency for conditional probabilities

$$\tilde{p}(x|y) = \frac{n(x, y)}{n_y}$$

Conditional probabilities can also be estimated from relative frequencies.

$n(x, y)$ here is the number of common occurrences of the events x and y.

n_y is the number of occurrences of the event y.

It applies: $n_y = \sum_{x'} n(x', y)$

Relative frequency for conditional probabilities

- Random event x: Word occurrence is a certain word
- Random event y: Word occurrence is in email of a certain category, e.g. HAM or SPAM (HAM = " no spam ")
- $n(x, y)$: Number of word occurrences in emails of a category in the corpus

word	n(word, HAM)	$\tilde{p}(\text{word}\|\text{HAM})$	n(word, SPAM)	$\tilde{p}(\text{word}\|\text{SPAM})$
meet				
deadline				
single				
...				

Relative frequency for conditional probabilities

- Random event x: Word occurrence is a certain word
- Random event y: Word occurrence is in email of a certain category, e.g. HAM or SPAM (HAM = " no spam ")
- $n(x, y)$: Number of word occurrences in emails of a category in the corpus

word	n(word, HAM)	$\tilde{p}(\text{word}\mid\text{HAM})$	n(word, SPAM)	$\tilde{p}(\text{word}\mid\text{SPAM})$
meet	1	$\frac{1}{9} \approx 0.111$	1	$\frac{1}{6} \approx 0.167$
deadline	2	$\frac{2}{9} \approx 0.222$	0	0
single	0	0	1	$\frac{1}{6} \approx 0.167$
...				

Probability for word sequence

- So far we have only expressed and estimated probabilities of single words.
- How can we calculate the probabilities of whole texts (e.g. emails)?
- Application of conditional probability:

$$P(w_1, w_2, \ldots, w_n)$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \ldots P(w_n|w_1 \ldots w_{n-1})$$

- $\Rightarrow$ does not really solve the problem, because $P(w_n|w_1 \ldots w_{n-1})$ can not be well estimated

Independence assumption: Bag of Words

- One solution: we make the statistical assumption that every word is independent of the occurrence of other words.
- This is also called bag-of-words (BOW) assumption, because the order of words becomes irrelevant.

$$P(w_1, w_2, \ldots, w_n)$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\ldots P(w_n|w_1 \ldots w_{n-1})$$

$$\underset{\text{indep.}}{=} P(w_1)P(w_2)P(w_3)\ldots P(w_n)$$

Conditional independence

- For many machine-learning algorithms, **conditional independence** is the central concept:
  If the value of a random variable y is known, random variables $x_1, \ldots, x_n$ are independent
- Middle ground between:
  - no independence
  - independence of all random variables
- In our case:

$$P(w_1, w_2, \ldots, w_n|\text{SPAM})$$

$$\underset{\text{cond. indep.}}{=} P(w_1|\text{SPAM})P(w_2|\text{SPAM})P(w_3|\text{SPAM}) \ldots P(w_n|\text{SPAM})$$

Naive Bayes Classifier

Task

- Given a training corpus:
- Decide whether new (unseen) emails should be assigned to the category HAM or SPAM:

Decision criterion

- Given the content of the email, which category is more likely SPAM or HAM?

$$P(HAM|text) > P(SPAM|text)$$

- Why isn't the decision criterion:

$$P(text|HAM) > P(text|SPAM)$$

?

Bayes rule

$$P(HAM|text) = \frac{P(text|HAM) * P(HAM)}{P(text)}$$

- $P(text|HAM)$: conditional BOW probability
- $P(HAM)$: Prior probability that an email is assigned to the category HAM (if the content of the email is not known). Estimation:

$$\tilde{p}(HAM) = \frac{\text{number of HAM-Mails}}{\text{number of all Mails}}$$

- $P(text)$: BOW probability of the content of the email without the category being known

Decision criterion

$$\text{Email is HAM}$$

$$\Leftrightarrow$$

$$P(HAM|text) > P(SPAM|text)$$

$$\Leftrightarrow$$

$$\frac{P(HAM|text)}{P(SPAM|text)} > 1$$

$$\Leftrightarrow$$

Decision criterion

Email is HAM

$$\Leftrightarrow$$

$$P(HAM|text) > P(SPAM|text)$$

$$\Leftrightarrow$$

$$\frac{P(HAM|text)}{P(SPAM|text)} > 1$$

$$\Leftrightarrow$$

$$\frac{\cancel{\frac{1}{P(text)}} P(text|HAM) * P(HAM)}{\cancel{\frac{1}{P(text)}} P(text|SPAM) * P(SPAM)} > 1$$

What is a decision rule for more than two categories?

Example(temporarily)

- $\tilde{p}(HAM) = \frac{3}{5}$
- $\tilde{p}(SPAM) = \frac{2}{5}$
- $p(\text{hot stock for}|HAM)$

$$= \tilde{p}(\text{hot}|HAM)\tilde{p}(\text{stock}|HAM)\tilde{p}(\text{for}|HAM) = \ldots$$

- $p(\text{hot stock for}|SPAM)$

$$= \tilde{p}(\text{hot}|SPAM)\tilde{p}(\text{stock}|SPAM)\tilde{p}(\text{for}|SPAM) = \ldots$$

- ...

Example(temporarily)

- $\tilde{p}(HAM) = \frac{3}{5}$
- $\tilde{p}(SPAM) = \frac{2}{5}$
- $p(\text{hot stock for}|HAM)$

$$= \tilde{p}(\text{hot}|HAM)\tilde{p}(\text{stock}|HAM)\tilde{p}(\text{for}|HAM) = \frac{0 \cdot 0 \cdot 1}{9 \cdot 9 \cdot 9} = 0$$

- $p(\text{hot stock for}|SPAM)$

$$= \tilde{p}(\text{hot}|SPAM)p(\text{stock}|SPAM)\tilde{p}(\text{for}|SPAM) = \frac{2 \cdot 1 \cdot 0}{6 \cdot 6 \cdot 6} = 0$$

- Problem: Decision criterion is not defined $\left(\frac{0}{0}\right)$

Add-1 smooting

Add-1 smooting (Laplace smoothing)

$$\tilde{p}(w) = \frac{n(w) + 1}{N + V}$$

(V = number of possible words; N = number of tokens)

... is optimal if the uniform distribution is most likely is what is rarely the case in Textcorpora $\Rightarrow$ Zipf's distribution

... therefore, overestimates the probability of unseen words.

Add-λ smoothing

reduces the amount of smoothing

Add-λ smoothing

$$\tilde{p}(w) = \frac{n(w) + \lambda}{N + V\lambda}$$

Add-λ smoothing for conditional probabilities

$$\tilde{p}(w|y) = \frac{n(w, y) + \lambda}{n_y + v\lambda}$$

Example (with Add-1 smoothing)

- $\tilde{p}(HAM) = \frac{3}{5}$, $\tilde{p}(SPAM) = \frac{2}{5}$
- Vocabulary contains $v = 10$ different words
- $p(\text{hot stock for}|HAM) = \tilde{p}(\text{hot}|HAM)\tilde{p}(\text{stock}|HAM)\tilde{p}(\text{for}|HAM)$

$$= \frac{(0+1)\cdot(0+1)\cdot(1+1)}{(9+10)\cdot(9+10)\cdot(9+10)} \approx 0.00029$$

- $p(\text{hot stock for}|SPAM) = \tilde{p}(\text{hot}|SPAM)\tilde{p}(\text{stock}|SPAM)\tilde{p}(\text{for}|SPAM)$

$$= \frac{(2+1)\cdot(1+1)\cdot(0+1)}{(6+10)\cdot(6+10)\cdot(6+10)} \approx 0.00146$$

- $\frac{P(text|HAM)*P(HAM)}{P(text|SPAM)*P(SPAM)} = \frac{0.00029\cdot 0.6}{0.00146\cdot 0.4} \approx 0.298 \Rightarrow$ Category?

Example (with Add-1 smoothing)

- $\tilde{p}(HAM) = \frac{3}{5}$, $\tilde{p}(SPAM) = \frac{2}{5}$
- Vocabulary contains $v = 10$ different words
- $p(\text{hot stock for}|HAM) = \tilde{p}(\text{hot}|HAM)\tilde{p}(\text{stock}|HAM)\tilde{p}(\text{for}|HAM)$

$$= \frac{(0+1)\cdot(0+1)\cdot(1+1)}{(9+10)\cdot(9+10)\cdot(9+10)} \approx 0.00029$$

- $p(\text{hot stock for}|SPAM) = \tilde{p}(\text{hot}|SPAM)\tilde{p}(\text{stock}|SPAM)\tilde{p}(\text{for}|SPAM)$

$$= \frac{(2+1)\cdot(1+1)\cdot(0+1)}{(6+10)\cdot(6+10)\cdot(6+10)} \approx 0.00146$$

- $\frac{P(text|HAM)*P(HAM)}{P(text|SPAM)*P(SPAM)} = \frac{0.00029\cdot 0.6}{0.00146\cdot 0.4} \approx 0.298 < 1 \Rightarrow$ Email is spam

Calculating with logarithms

- When multiplying many small probabilities (for example, all words in a long text), the result can quickly approach 0 and may not be correctly represented.
- That's why you always avoid the multiplication of probabilities.
- Instead, use the sum of the logarithmized probabilities.
- $\log(a \cdot b \cdot c \cdot \ldots) = \log(a) + \log(b) + \log(c) + \ldots$
- Example:

$$0.0001 * 0.001 * 0.00001 * 0.01 = 0.00000000000001$$

$\log_{10}(0.0001 * 0.001 * 0.00001 * 0.01) =$

Calculating with logarithms

- When multiplying many small probabilities (for example, all words in a long text), the result can quickly approach 0 and may not be correctly represented.
- That's why you always avoid the multiplication of probabilities.
- Instead, use the sum of the logarithmized probabilities.
- $\log(a \cdot b \cdot c \cdot \ldots) = \log(a) + \log(b) + \log(c) + \ldots$
- Example:

$$0.0001 * 0.001 * 0.00001 * 0.01 = 0.00000000000001$$

$$\log_{10}(0.0001 * 0.001 * 0.00001 * 0.01) = -4 + (-3) + (-5) + (-2) = -14$$

- $\log(\frac{a}{b}) = ?$

Decision rule with logarithms

- The logarithm is increasing monotonically, i.e. we can apply it to inequalities on both sides.
- The decision rule is now:

$$P(HAM|text) > P(SPAM|text)$$

$$\Leftrightarrow$$

$$\log P(HAM|text) > \log P(SPAM|text)$$

$$\Leftrightarrow$$

Decision rule with logarithms

- The logarithm is increasing monotonically, i.e. we can apply it to inequalities on both sides.
- The decision rule is now:

$$P(HAM|text) > P(SPAM|text)$$

$$\Leftrightarrow$$

$$\log P(HAM|text) > \log P(SPAM|text)$$

$$\Leftrightarrow$$

$$\log P(HAM|text) - \log P(SPAM|text) > 0$$

$$\Leftrightarrow$$

$$\log P(text|HAM) + \log P(HAM) - \log P(text|SPAM) - \log P(SPAM) > 0$$

- The quotient of the probabilities of two complementary events is also called **Odds**.
- The logarithm of this quotient is called **Log-Odds**.

Naive Bayes with other distributions

- Depending on the problem, the distribution $P(X|category)$ can take various forms.
- In the case just discussed, the distribution is the **multinomial distribution** (probability that with text of length n exactly the observed numbers of words occur)
- If the observed values are floating point values (e.g., values from a meter), one can e.g. Use **Gauss Distributions**.
  $\Rightarrow$ Smoothing is also important here (for example, what variance should be assumed for a category with little data?)
- For machine-learning software (such as Scikit-learn), you can choose the type of distribution as a hyper parameter.

Unknown words in the test data

- It may be that words appear in the test data that did not occur in the training data.
- The possible values of the random variable were chosen based on the training data, i.e. the probability of the new words is not defined.
- Two common solutions:
  - Words that do not occur in the training data are ignored ($\Rightarrow$ Test documents are getting shorter)
  - Words that are rarely in the training data (for example, 1-2 times) or do not occur at all, (in training and testing) are replaced with a placeholder `<UNK>`.
- Advantages and disadvantages of the two methods?

Implementation

Training or test instance

In our case:

- Features = words (Tokens)
- Label
  - Binary classification: HAM (True) vs SPAM (False)
  - Multi-Class Classification (Exercise Sheet): String for category(`"work"`, `"social"`, `"promotions"`, `"spam"`, ...)

```
class DataInstance:
    def __init__(self, feature_counts, label):
        self.feature_counts = feature_counts
        self.label = label
#...
```

Training or test set

- Amount of possible feature values is e.g. important for smoothing.
- Sanity-check: What accuracy would have prediction of the most common category?
- Some learning algorithms require several training iterations between which the training set should be re-permuted (mixed).

```
class Dataset:
    def __init__(self, instance_list, feature_set):
        self.instance_list = instance_list
        self.feature_set = feature_set
    def most_frequent_sense_accuracy(self):
        # ...
    def shuffle(self):
        # ...
```

Classifier

What information do we need to create the Naive-Bayes model?

- ...

Classifier

What information do we need to create the Naive-Bayes model?

- For the estimation of $P(w|HAM)$ or $P(w|SPAM)$
  - $n(w, HAM)$ or $n(w, SPAM)$:
    One dictionary each, which maps each word to its frequency in the respective category.
  - n_{HAM} or n_{SPAM}:
    The number of word occurrences per category
    (can be summed up from the values of the dictionaries)
  - For smoothing: Parameters λ and size of the vocabulary v
- For the estimation of $P(HAM)$ or $P(SPAM)$
  - In each case the number of training emails per category.

Classifier: constructor

```
def __init__(self, positive_word_to_count, negative_word_to_count,\
        positive_counts, negative_counts, vocabsize, smoothing):
    # n(word, HAM) and n(word, SPAM)
    self.positive_word_to_count = positive_word_to_count
    self.negative_word_to_count = negative_word_to_count

    # n_HAM and n_SPAM
    self.positive_total_wordcount = \
        sum(positive_word_to_count.values())
    self.negative_total_wordcount = \
        sum(negative_word_to_count.values())

    self.vocabsize = vocabsize

    # P(HAM) and P(SPAM)
    self.positive_prior = \
        positive_counts / (positive_counts + negative_counts)
    self.negative_prior = \
        negative_counts / (positive_counts + negative_counts)

    self.smoothing = smoothing
```

Classifier: Overview

```
class NaiveBayesWithLaplaceClassifier:
    def log_probability(self, word, is_positive_label):
        # ...
    def log_odds(self, feature_counts):
        # ...
    def prediction(self, feature_counts):
        # ...
    def prediction_accuracy(self, dataset):
        # ...
    def log_odds_for_word(self, word):
        # ...
    def features_for_class(self, is_positive_class, topn=10):
        # ...
```

Calculation of $P(w|HAM)$ or $P(w|SPAM)$

Probability estimation...

- ... smoothed
- ... is returned logarithmized

```
def log_probability(self, word, is_positive_label):
    if is_positive_label:
        wordcount = self.positive_word_to_count.get(word, 0)
        total = self.positive_total_wordcount
    else:
        wordcount = self.negative_word_to_count.get(word, 0)
        total = self.negative_total_wordcount
    return math.log(wordcount + self.smoothing) \
        - math.log(total + self.smoothing * self.vocabsize)
```

Calculation of Log-Odds

- What is calculated in the two sums?

```
def log_odds(self, feature_counts):
    # language model probability
    pos_logprob = sum([ count * self.log_probability(word, True) \
        for word, count in feature_counts.items()])
    # prior probability
    pos_logprob += math.log(self.positive_prior)
    # same for negative case
    neg_logprob = sum([ count * self.log_probability(word, False) \
        for word, count in feature_counts.items()])
    neg_logprob += math.log(self.negative_prior)
    return pos_logprob - neg_logprob
```

Applying the classifier, test accuracy

- Prediction
  - Apply the model to the feature counts of a test instance
  - Prediction of a category (HAM/True or SPAM/False) according to the decision rule

```
def prediction(self, feature_counts):
    # ...
```

- Calculation of test accuracy
  - First, prediction for all instances of the dataset
  - Then compare with the correct category label

```
def prediction_accuracy(self, dataset):
    # ...
```

Multi-class classification

Multi-class classification

- Extension: Classifier distinguishes n different categories($n \geq 2$)
- $\Rightarrow$ Exercise sheet
- Deciosion rule: choose category c^*, that maximizes probability $p(c^*|text)$.

$$c^* = \arg\max_c p(c|text)$$

- $\arg\max_x f(x)$ selects a value x (from the definition set) for which the function value $f(x)$ is maximal.
- By applying the calculation rules, the conditional independence assumption, and our estimation method (Laplace):

$$c^* = \arg\max_c p(c)p(text|c)$$

$$= \arg\max_c \log[p(c)] + \sum_{w \in text} \log[\tilde{p}(w|c)]$$

Multi-class classification

- Decision rule: choose category c^*, that maximizes probability $p(c^*|text)$.

$$c^* = \arg\max_c p(c|text)$$

- Does the following implication apply?

$$c^* = \arg\max_c p(c|text) \Rightarrow \frac{p(c^*|text)}{1 - p(c^*|text)} \geq 1$$

- Does the following implication apply?

$$\frac{p(c^*|text)}{1 - p(c^*|text)} > 1 \Rightarrow c^* = \arg\max_c p(c|text)$$

Multi-class classification

- Does the following implication apply?

$$c^* = \arg\max_c p(c|text) \Rightarrow \frac{p(c^*|text)}{1 - p(c^*|text)} \geq 1$$

No. For 3 or more categories, the most likely category may be WK $p(c^*|text) < 0.5$ *and the Odds are* < 1.

- Does the following implication apply?

$$\frac{p(c^*|text)}{1 - p(c^*|text)} > 1 \Rightarrow c^* = \arg\max_c p(c|text)$$

Yes. If the most likely category odds has > 1, *the WK* $p(c^*|text) > 0.5$, *and all other categories must have a smaller WK.*

Multi-classes Naive Bayes: Implementation

- To calculate the values $\tilde{p}(w|c)$, we need the word frequencies per class $n(w,c)$
  Solution: Dictionary
  (str,str) $\rightarrow$ int
- For the priors $p(c)$ we need the number of instances per class:
  str $\rightarrow$ int
- Also the vocabulary size and the smoothing parameter

```
class NaiveBayesClassifier:
    def __init__(self, word_and_category_to_count, \
        category_to_num_instances, vocabsize, smoothing):
        # ...
```

Log odds per word

$\Rightarrow$ Exercise sheet

- The log odds for a category c can also be calculated for only one word (instead of one whole document).
- Begin with $\log \frac{p(c|w)}{1-p(c|w)}$ and apply the calculation rules

$$\log \frac{p(c|w)}{1 - p(c|w)} = \dots$$

$$= \log[\tilde{p}(w|c)] + \log[p(c)] - \log[\sum_{c' \neq c} \tilde{p}(w|c')p(c')]$$

- The log odds per word indicate how strongly a word indicates the respective category
- You can then sort all words based on their log odds, and get an idea of what the model has learned (i.e., what's important for the model)

Train and evaluate a classifier

In order to train and evaluate a classifier, we need 3 datasets:

1. Training data: On this data, the model estimates its parameters automatically (e.g., word probabilities and category priors).
2. Development data: On this data, various model architectures and hyper-parameters[1] can be compared.
   What, for example in our case?
3. Test data: An estimate of how well the model works on further unseen data can be obtained on this data, **after the development data finally determined a model architecture**.
   - Depending on the nature of the data (domain etc), this estimate can deviate very much from reality.
   - The estimate may also be very unreliable depending on the amount of test data ($\Rightarrow$ significance tests).
   - **Why can not we use the performance on the development data?**

[1] Parameters that are not automatically learned.

Summary

- Probability calculation
  - Theorem of Bayes
  - Conditional independence
- Naive Bayes classificator
  - Decision rule, and “flip” the formula by using theorem of Bayes
  - Smoothing the probabilities
  - Log-Odds
- Questions?