

Classification via counting

A model for the Classification task constructs a probability distribution $\hat{\mathbf{y}} = p(\mathbf{y}|\mathbf{x})$

- Given feature vector \mathbf{x}
- Construct vector $\hat{\mathbf{y}}$ (of length $|C|$, where C are the distinct values for the target)
- Whose elements are probabilities: $\hat{\mathbf{y}}_c$ is the probability that \mathbf{x} is in class c

Notation abuse alert:

- Subscripts of vectors should be integers rather than class names
- So technically we should write $\hat{\mathbf{y}}_{\#c}$ where $\#c$ is the integer index of class named c

This sounds difficult at first glance.

Let's start with something simpler: counting.

We will show to to construct this probability using nothing more than counting the features and targets of the training set!

From counting to probability

We introduce the topic by assuming all our variables (features and target) are discrete.

We will subsequently adapt this to continuous variables.

First, let's compute the distribution of target classes.

Let $\mathbf{X} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) | 1 \leq i \leq m\}$ be our m training examples.

Then

$$count_{\mathbf{y}=\mathbf{y}'} = |\{i | \mathbf{y}^{(i)} = \mathbf{y}'\}|$$

is the number of training examples with target \mathbf{y}' .

We can easily convert this into an unconditional probability

$$p(\mathbf{y} = \mathbf{y}') = \frac{count_{\mathbf{y}=\mathbf{y}'}}{m}$$

We can similarly compute the *joint* probability of any two features.

First we count the co-occurrences of the two variables

$$count_{\mathbf{x}_j=\mathbf{x}'_j, \mathbf{x}_k=\mathbf{x}'_k} = \left| \{i | \mathbf{x}_j^{(i)} = \mathbf{x}'_j, \mathbf{x}_k^{(i)} = \mathbf{x}'_k\} \right|$$

And the the joint probability is

$$p(\mathbf{x}_j = \mathbf{x}'_j, \mathbf{x}_k = \mathbf{x}'_k) = \frac{count_{\mathbf{x}_j=\mathbf{x}'_j, \mathbf{x}_k=\mathbf{x}'_k}}{m}$$

Our illustration is with two features but the notation generalizes for

- Any number of variables
- Any kind of variables: feature or target

Finally, we can define *conditional* probability

$$p(\mathbf{y} = \mathbf{y}' | \mathbf{x} = \mathbf{x}') = \frac{p(\mathbf{y} = \mathbf{y}', \mathbf{x} = \mathbf{x}')}{p(\mathbf{x} = \mathbf{x}')}$$

That is, the conditional probability

- Is the joint probability
- As a fraction, relative to the unconditional probability of $\mathbf{x} = \mathbf{x}'$

Bayes theorem

The key for converting counts (really, associated probabilities) to predictions lies in Bayes Theorem.

Bayes Theorem relates conditional and unconditional probabilities.

Bayes Theorem

$$p(\mathbf{y} = \mathbf{y}' | \mathbf{x} = \mathbf{x}') = \frac{p(\mathbf{x} = \mathbf{x}' | \mathbf{y} = \mathbf{y}') * p(\mathbf{y} = \mathbf{y}')}{p(\mathbf{x} = \mathbf{x}')}$$

Let's think about Bayes Theorem in terms of our classification task:

- The left hand side is our prediction for the class probabilities, given the features
- The right hand side involves
 - The conditional probability of seeing examples with features \mathbf{x}' and target \mathbf{y}' .
 - The unconditional probability of seeing examples with label \mathbf{y}'
 - The unconditional probability of seeing examples with feature vector \mathbf{x}' .

All these elements can be obtained by counting (and filtering) the training set !

Hence, we can build an extremely simple classifier using nothing more than counting.

Posterior, Prior Probability, Evidence

Let's break down the parts of Bayes theorem and give them some names:

- $p(\mathbf{y} = \mathbf{y}' | \mathbf{x} = \mathbf{x}')$: *posterior probability*
 - Our prediction
 - This is the probability distribution of \mathbf{y} *conditional* on the features being \mathbf{x}
- $p(\mathbf{y} = \mathbf{y}')$: *prior probability*
 - This is the unconditional distribution of \mathbf{y}
- $p(\mathbf{x} = \mathbf{x}' | \mathbf{y} = \mathbf{y}')$: *likelihood*
 - Given that $\mathbf{y} = \mathbf{y}'$, what is the probability that $\mathbf{x} = \mathbf{x}'$?
 - This is the counting part: how often does the label \mathbf{y}' occur when the features are \mathbf{x}' ?
- $p(\mathbf{x} = \mathbf{x}')$: *evidence*
 - How often do we see the features \mathbf{x}' ?

We can re-state Bayes Theorem as

$$\text{posterior} = \frac{\text{prior} * \text{likelihood}}{\text{evidence}}$$

That is:

- Starting from an uninformed *prior* distribution of \mathbf{y}
- Derive a conditional *posterior* distribution (i.e., informed by *evidence* \mathbf{x}) by updating via the *likelihood* of seeing \mathbf{x}, \mathbf{y} together.

Proof of Bayes Theorem

$$\begin{aligned} p(\mathbf{y} = \mathbf{y}' | \mathbf{x} = \mathbf{x}') &= \frac{p(\mathbf{y}=\mathbf{y}', \mathbf{x}=\mathbf{x}')}{p(\mathbf{x}=\mathbf{x}')} && \text{(def. of conditional probability)} \\ &= \frac{p(\mathbf{y}=\mathbf{y}', \mathbf{x}=\mathbf{x}')}{p(\mathbf{x}=\mathbf{x}')} * \frac{\frac{1}{p(\mathbf{y}=\mathbf{y}')}}{\frac{1}{p(\mathbf{y}=\mathbf{y}')}} && \text{(multiply by identity)} \\ &= \frac{p(\mathbf{x}=\mathbf{x}' | \mathbf{y}=\mathbf{y}')}{p(\mathbf{x}=\mathbf{x}')} * \frac{1}{\frac{1}{p(\mathbf{y}=\mathbf{y}')}} && \text{(def. of conditional probability)} \\ &= \frac{p(\mathbf{x}=\mathbf{x}' | \mathbf{y}=\mathbf{y}')}{p(\mathbf{x}=\mathbf{x}')} * p(\mathbf{y} = \mathbf{y}') \end{aligned}$$

Length of \mathbf{x} is n

Remember that \mathbf{x} is a vector, so that $p(\mathbf{x} = \mathbf{x}' \mid \mathbf{y} = \mathbf{y}')$ is a *joint* probability of n terms

$$p(\mathbf{x}_1 = \mathbf{x}'_1, \mathbf{x}_2 = \mathbf{x}'_2, \dots, \mathbf{x}_n = \mathbf{x}'_n \mid \mathbf{y} = \mathbf{y}')$$

We can obtain this by counting (as described above)

- Let $|\mathbf{x}_j|$ denote the number of distinct values for the j^{th} feature
- There are

$$\prod_{1 \leq j \leq n} |\mathbf{x}_j|$$

potential combinations for \mathbf{x}

That's a lot of counting !

More importantly, it's a lot of parameters to remember (i.e, size of Θ is big).

We need a short-cut.

The Naive part of Naive Bayes

We will assume that each feature is *conditionally* independent of one another

$$p(\mathbf{x}_j = \mathbf{x}'_j, \mathbf{x}_k = \mathbf{x}'_k, | \mathbf{y} = \mathbf{y}') = p(\mathbf{x}_j = \mathbf{x}'_j | \mathbf{y} = \mathbf{y}') * p(\mathbf{x}_k = \mathbf{x}'_k | \mathbf{y} = \mathbf{y}')$$

That is

- \mathbf{x}_j and \mathbf{x}_k are **not** independent unconditionally
- They **are** independent *conditional* on $\mathbf{y} = \mathbf{y}'$

Think of \mathbf{x}_j and \mathbf{x}_k being correlated through their individual relationships with \mathbf{y} .

Excluding that mutual dependence, they may be uncorrelated.

Generalizing the assumption to feature vectors \mathbf{x} of length n :

$$p(\mathbf{x} = \mathbf{x}' | \mathbf{y} = \mathbf{y}') = \prod_{i=1}^n p(\mathbf{x}_i = \mathbf{x}'_i | \mathbf{y} = \mathbf{y}')$$

That is

- The joint conditional probability of the vector of length n
- Is **assumed** to be the product of the individual conditional probabilities of each element of the vector.

This assumption is probably not true but

- Makes $p(\mathbf{x} = \mathbf{x}' | \mathbf{y} = \mathbf{y}')$ very easy to compute
 - Don't have to compute it for possible combination of values for \mathbf{x}
- Uses few parameters
- May be close enough

Thus the "naive" assumption has many benefits !

What about computing the *unconditional* $p(\mathbf{x} = \mathbf{x}')$?

We can obtain this from conditional probabilities as well

$$p(\mathbf{x} = \mathbf{x}') = \sum_{c \in C} p(\mathbf{x} = \mathbf{x}' | \mathbf{y} = c) * p(\mathbf{y} = c)$$

That is, the unconditional probability follows from the

- Conditional probability given \mathbf{y}
- Weighted by the probability $p(\mathbf{y})$ for each possible value of \mathbf{y}

This follows from the definition of conditional probability.

What this means is that the only parameters we need to remember are

- The unconditional probabilities $p(\mathbf{y})$
 - Depends on number of classes $|C|$
- Probabilities conditional on \mathbf{y} : $p(\mathbf{x}|\mathbf{y})$
 - Depends on length of \mathbf{x} : n

Example

Here is a hypothetical trading example for equities

- There are two categorical features
 - Valuation: possible values {Rich, Cheap}
 - Is the current stock price expensive (Rich) or inexpensive (Cheap) ?
 - Yield: possible values {High, Low}
 - Is the dividend yield of the stock desirable (High) or undesirable (Low) ?
- Target: An Action with possible values {Long, Short, Neutral}
 - What should our position be ?

We are given a number of examples (on which to train).

Our Classification task is

- Given an equity (test example) with values for the two features Valuation and Yield
- Decide what our position (Long/Short/Neutral) should be

Here are our training examples

```
In [4]: d_df = pd.read_csv("valuation_yield_action.csv")
target_name = "Action"
d_df
```

Out[4]:

	Valuation	Yield	Action
0	Cheap	High	Long
1	Cheap	High	Long
2	Cheap	High	Long
3	Cheap	High	Neutral
4	Rich	Low	Short
5	Rich	Low	Short
6	Rich	Low	Short
7	Rich	Low	Short
8	Rich	Low	Neutral
9	Cheap	Low	Neutral
10	Cheap	Low	Long
11	Cheap	Low	Short
12	Cheap	Low	Neutral
13	Rich	High	Long
14	Rich	High	Short
15	Rich	High	Neutral
16	Fair	Low	Long
17	Fair	Low	Short
18	Fair	Low	Neutral
19	Fair	High	Long
20	Fair	High	Short
21	Fair	High	Neutral
22	Cheap	Low	Long
23	Fair	Low	Short
24	Fair	High	Long

And a quick look at the data, sliced by Action

```
In [5]: grouped_by_target = d_df.groupby(target_name)
        for gp in grouped_by_target.groups.keys():
            print(gp, "\n")
            print(grouped_by_target.get_group(gp).head())
            print("\n\n")
```

Long

	Valuation	Yield	Action
0	Cheap	High	Long
1	Cheap	High	Long
2	Cheap	High	Long
10	Cheap	Low	Long
13	Rich	High	Long

Neutral

	Valuation	Yield	Action
3	Cheap	High	Neutral
8	Rich	Low	Neutral
9	Cheap	Low	Neutral
12	Cheap	Low	Neutral
15	Rich	High	Neutral

Short

	Valuation	Yield	Action
4	Rich	Low	Short
5	Rich	Low	Short
6	Rich	Low	Short
7	Rich	Low	Short
11	Cheap	Low	Short

Looks like we

- Go Long if the stock is Cheap (Valuation) and High (Yield)
- Go Short if the stock is Rich (expensive Valuation) and Low (Yield)

Here's the empirical distribution of the training examples

```
In [6]: d_df["dummy"] = 1 # Need to aggregate on something
t = d_df.pivot_table(index=target_name, columns=["Valuation", "Yield"], values
="dummy", aggfunc=["count"],
                    margins=True)

t
```

Out[6]:

	count						
Valuation	Cheap		Fair		Rich		All
Yield	High	Low	High	Low	High	Low	
Action							
Long	3.0	2.0	2.0	1.0	1.0	NaN	9
Neutral	1.0	2.0	1.0	1.0	1.0	1.0	7
Short	NaN	1.0	1.0	2.0	1.0	4.0	9
All	4.0	5.0	4.0	4.0	3.0	5.0	25

This gives us everything we need for the Naive Bayes algorithm

- These are counts
- We can easily turn the counts into unconditional probabilities by dividing by total number of examples
- Will leave them as counts for now

Let's parse this table:

- Columns: $count_{y|x}$
 - A column (defined by concrete values for each of the two attributes)
 - Defines a distribution over the target (Action)
- Column Sum: $count_{\mathbf{x}} = \sum_{a \in \text{Action}} count_{\mathbf{x}|a}$
 - Total number of examples with attribute pair \mathbf{x}
- Rows: $count_{\mathbf{x}|y}$
 - A row (defined by a concrete value for the Action)
 - Defines a distribution over the attributes pairs for which this action is taken
- Row sums: $count_a = \sum_{\mathbf{x}} count_{\mathbf{x}|a}$
 - Total number of examples with Action a

Let's simplify the table by looking at the marginal with respect to each attribute

- Distribution over a single attribute rather than the pair

First, by Valuation

```
In [7]: # Single feature (Valuation), rather than pair  
d_df.drop(columns=["dummy"]).pivot_table(index=target_name, columns=["Valuation",  
n"], aggfunc=["count"],  
fill_value=0, margins=True)
```

Out[7]:

count				
Yield				
Valuation	Cheap	Fair	Rich	All
Action				
Long	5	3	1	9
Neutral	3	2	2	7
Short	1	3	5	9
All	9	8	8	25

And by Yield


```
In [8]: # Single feature (Yield), rather than pair  
d_df.drop(columns=["dummy"]).pivot_table(index=target_name, columns=["Yield"],  
aggfunc=["count"],  
fill_value=0, margins=True)
```

Out[8]:

count			
Valuation			
Yield	High	Low	All
Action			
Long	6	3	9
Neutral	3	4	7
Short	2	7	9
All	11	14	25

And the target (Action) distribution

```
In [9]: t.loc[:, idx["count", "All", :]]
```

Out[9]:

count	
Valuation	All
Yield	
Action	
Long	9
Neutral	7
Short	9
All	25

Here is the target distribution as probabilities rather than counts

```
In [10]: num_examples = t.loc["All", idx["count", "All", :]][0]

print("There are {e:d} training examples".format(e=int(num_examples)) )

# Class probabilities
t.loc[:, idx["count", "All", :]]/t.loc["All", idx["count", "All", :]]
```

There are 25 training examples

Out[10]:

count	
Valuation	All
Yield	
Action	
Long	0.36
Neutral	0.28
Short	0.36
All	1.00

Features with continuous rather than discrete values

The counting approach works well when a feature's values are discrete rather than continuous

- countable

The simplest way to deal with a continuous feature \mathbf{x}_j

- make it discrete by replacing them with categorical indicator features

For example

- Divide the range of values into two intervals defined by threshold value t_j

$$\mathbf{I}_{\mathbf{x}_j < t_j}$$

- Divide the range of values into multiple intervals ("buckets") by a sequence of thresholds defining boundaries

$$\mathbf{I}_{t_{j,l-1} \leq \mathbf{x}_j \leq t_{j,l}}$$

The thresholds are a *hyper-parameters*

- not parameters that are optimized with the Loss function
- The Fine-tuning part of the Recipe is the place where we search for alternative values

Unfortunately

- the *ordering relationship* between continuous values is lost when converting them into binary indicators

We mention this technique

- because we have seen it used in Decision Trees
- but there are alternatives that preserve the features as continuous rather than making them discrete

Beyond counting

The assumption of conditional independence (the Naive part) makes prediction based on Bayes Theorem practical.

The most obvious way: less to count (and store)

- We only need the marginal probability of each features
- Rather than the join probability of each pair, triple, etc. of features
- The joint probability can be derived from the marginals under the assumption of conditional independence

For two features $\mathbf{x}_1, \mathbf{x}_2$

- we store the counts for each possible value (in the training set) of \mathbf{x}_1 and \mathbf{x}_2 individually (for each y)
- rather than a count for every possible pair of values $\mathbf{x}_1, \mathbf{x}_2$

number of stored values

independence $|\text{distinct}(\mathbf{x}_1)| + |\text{distinct}(\mathbf{x}_2)|$

NO independence $|\text{distinct}(\mathbf{x}_1)| * |\text{distinct}(\mathbf{x}_2)|$

where $\text{distinct}(x_j) = \{v | \mathbf{x}_j^{(i)} = v \text{ for some example } i\}$

The less obvious answer:

It facilitates *extrapolation* beyond training examples.

Suppose

- there is a non-training example with concrete values $\mathbf{x}'_1, \mathbf{x}'_2$ for features $\mathbf{x}_1, \mathbf{x}_2$
- but there is **no** training example i where $\mathbf{x}_1^{(i)} = \mathbf{x}'_1, \mathbf{x}_2^{(i)} = \mathbf{x}'_2$

In that case, a joint probability obtained from counting yields

$$p(\mathbf{x}_1 = \mathbf{x}'_1, \mathbf{x}_2 = \mathbf{x}'_2, |\mathbf{y} = \mathbf{y}') = 0$$

But a joint probability derived from the assumption of conditional independence yields

$$p(\mathbf{x}_1 = \mathbf{x}'_1, \mathbf{x}_2 = \mathbf{x}'_2, |\mathbf{y} = \mathbf{y}') = p(\mathbf{x}_1 = \mathbf{x}'_1 | \mathbf{y} = \mathbf{y}') * p(\mathbf{x}_2 = \mathbf{x}'_2 | \mathbf{y} = \mathbf{y}')$$

Thus, we can predict for test examples not represented in the training dataset.

Representing the marginal probability by a function

But this still does not completely solve the problem of extrapolation beyond training data.

The issue we previously raised was when

- the value for more than one feature did not co-occur in the training examples.

But what about the case where there are **no** training examples with the value \mathbf{x}'_1 for a *single* feature \mathbf{x}_1 ?

This is an illustration of the *zero frequency* problem.

It is especially problematic because

$$p(\mathbf{x}_1) = 0$$

can occur both in the numerator and the denominator

- divide by zero issue

Additive smoothing

[additive smoothing](https://en.wikipedia.org/wiki/Additive_smoothing)(https://en.wikipedia.org/wiki/Additive_smoothing),

A simple (but unsatisfying) solution to the division by zero and other zero frequency related problems

- Artificially inflate all counts by some parameter α .

This eliminates zero counts at the cost of biasing all counts.

Note that when converting counts to probabilities

- We have increased the count of each of the $|C|$ classes by α
- So the total count for the denominator is $m + |C| * \alpha$

Replace empirical distributions by functional forms

The more common way to address the zero frequency problem

- **assume** that the empirical marginal distribution for each feature follows some functional form
- For example
 - Normal distribution for continuous features
 - Bernoulli distribution for binary categorical variables
 - Multinomial distribution for categorical variables with more than two classes

One can fit the parameters of the assumed distribution (e.g., mean, frequency) from the empirical training data.

There is an additional benefit to using a functional form

- We can treating features with *continuous* values **directly**
- Rather than converting them into *discrete* values by "bucketing" with indicators

There is also a benefit of reduced storage

- since we only need to store parameters of the distribution rather than the entire empirical.

In `sklearn` 's implementation of Naive Bayes

- one must specify empirical distributions (?)

Assumption of conditional independence

This is a questionable assumption.

In its defense:

- If n (the number of features) is very large
 - The conditional independence assumption is more likely to hold.

```
In [11]: print("Done")
```

Done

