Classification via counting

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

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

Notation abuse alert:

- Subscripts of of vectors should be integers rather than class names
- So technically we should write $\backslash\hat{\mathbf{y}}_{\backslash\mathrm{loc}}c$ where $\backslash\mathrm{loc}c$ 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 $\X = \{(\x^{\ip}, \y^{\ip}) | 1 \le i \le m\}$ be our m training examples.

Then

$$\cnt\y = \y' = \left|\{i \mid \y^{\ip} = \y'\}\right|$$

is the number of training examples with target $\y'$.

We can easily convert this into an unconditional probability

$$\pr\y = \y' = \frac{\cnt\y = \y'}{m}$$

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

First we count the co-occurrences of the two variables

$$\backslash\text{cnt}\backslash\text{x}_j = \backslash\text{x}'_j, \backslash\text{x}_k = \backslash\text{x}'_k = \left| \{ i | \backslash\text{x}_j^{\backslash\text{ip}} = \backslash\text{x}'_j, \backslash\text{x}_k^{\backslash\text{ip}} = \backslash\text{x}'_k \} \right|$$

And the the joint probability is

$$\backslash\text{pr}\backslash\text{x}_j = \backslash\text{x}'_j, \backslash\text{x}_k = \backslash\text{x}'_k = \frac{\backslash\text{cnt}\backslash\text{x}_j = \backslash\text{x}'_j, \backslash\text{x}_k = \backslash\text{x}'_k}{m}$$

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

$$\backslash\mathrm{pr}\backslash\mathrm{y} = \backslash\mathrm{y}' | \backslash\mathrm{x} = \backslash\mathrm{x}' = \frac{\backslash\mathrm{pr}\backslash\mathrm{y} = \backslash\mathrm{y}', \backslash\mathrm{x} = \backslash\mathrm{x}'}{\backslash\mathrm{pr}\backslash\mathrm{x} = \backslash\mathrm{x}'}$$

That is, the conditional probability

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

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

$$\backslash\mathrm{pr}\backslash\mathrm{y} = \backslash\mathrm{y}' | \backslash\mathrm{x} = \backslash\mathrm{x}' = \frac{\backslash\mathrm{pr}\backslash\mathrm{x} = \backslash\mathrm{x}' | \backslash\mathrm{y} = \backslash\mathrm{y}' * \backslash\mathrm{pr}\backslash\mathrm{y} = \backslash\mathrm{y}'}{\backslash\mathrm{pr}\backslash\mathrm{x} = \backslash\mathrm{x}'}$$

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 $\backslash\mathbf{x}'$ and target $\backslash\mathbf{y}'$.
    - The unconditional probability of seeing examples with label $\backslash\mathbf{y}'$
    - The unconditional probability of seeing examples with feature vector $\backslash\mathbf{x}'$.

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:

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

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 \y
- Derive a conditional *posterior* distribution (i.e., informed by *evidence* \x) by updating via the *likelihood* of seeing \x, \y together.

Proof of Bayes Theorem

$$\backslash\mathrm{pr}\backslash\mathrm{y} = \backslash\mathrm{y}' | \backslash\mathrm{x} = \backslash\mathrm{x}' \quad = \quad \frac{\backslash\mathrm{pr}\backslash\mathrm{y}{=}\backslash\mathrm{y}', \backslash\mathrm{x}{=}\backslash\mathrm{x}'}{\backslash\mathrm{pr}\backslash\mathrm{x}{=}\backslash\mathrm{x}'} \qquad \text{(def. of conditional}$$

$$= \quad \frac{\backslash\mathrm{pr}\backslash\mathrm{y}{=}\backslash\mathrm{y}', \backslash\mathrm{x}{=}\backslash\mathrm{x}'}{\backslash\mathrm{pr}\backslash\mathrm{x}{=}\backslash\mathrm{x}'} * \frac{\frac{1}{\backslash\mathrm{pr}\backslash\mathrm{y}=\backslash\mathrm{y}'}}{\frac{1}{\backslash\mathrm{pr}\backslash\mathrm{y}=\backslash\mathrm{y}'}} \qquad \text{(multiply by identit}$$

$$= \quad \frac{\backslash\mathrm{pr}\backslash\mathrm{x}{=}\backslash\mathrm{x}' | \backslash\mathrm{y}{=}\backslash\mathrm{y}'}{\backslash\mathrm{pr}\backslash\mathrm{x}{=}\backslash\mathrm{x}'} * \frac{1}{\frac{1}{\backslash\mathrm{pr}\backslash\mathrm{y}=\backslash\mathrm{y}'}} \qquad \text{(def. of conditional}$$

$$= \quad \frac{\backslash\mathrm{pr}\backslash\mathrm{x}{=}\backslash\mathrm{x}' | \backslash\mathrm{y}{=}\backslash\mathrm{y}'}{\backslash\mathrm{pr}\backslash\mathrm{x}{=}\backslash\mathrm{x}'} * \backslash\mathrm{pr}\backslash\mathrm{y} = \backslash\mathrm{y}'$$

Length of $\backslash\mathbf{x}$ is n

Remember that $\backslash\mathrm{x}$ is a vector, so that $\backslash\mathrm{pr}\backslash\mathrm{x} = \backslash\mathrm{x}' \mid \backslash\mathrm{y} = \backslash\mathrm{y}'$ is a *joint* probability of n terms

$$\backslash\mathrm{pr}\backslash\mathrm{x}_1 = \backslash\mathrm{x}'_1, \backslash\mathrm{x}_2 = \backslash\mathrm{x}'_2, \ldots, \backslash\mathrm{x}_n = \backslash\mathrm{x}'_n \mid \backslash\mathrm{y} = \backslash\mathrm{y}'$$

We an obtain this by counting (as described above)

- Let $|\backslash\mathrm{x}_j|$ denote the number of distinct values for the j^{th} feature
- There are
$$\prod_{1 \le j \le n} |\backslash\mathrm{x}_j|$$
potential combinations for $\backslash\mathrm{x}$

That's a lot of counting !

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

The Naive part of Naive Bayes

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

$$\backslash\mathrm{pr}\backslash\mathbf{x}_j = \backslash\mathbf{x}'_j, \backslash\mathbf{x}_k = \backslash\mathbf{x}'_k, | \backslash\mathbf{y} = \backslash\mathbf{y}' = \backslash\mathrm{pr}\backslash\mathbf{x}_j = \backslash\mathbf{x}'_j | \backslash\mathbf{y} = \backslash\mathbf{y}'$$
$$* \backslash\mathrm{pr}\backslash\mathbf{x}_k = \backslash\mathbf{x}'_k | \backslash\mathbf{y} = \backslash\mathbf{y}'$$

That is

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

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

Excluding that mutual dependence, they may be uncorrelated.

Generalizing the assumption to feature vectors $\backslash\mathrm{x}$ of length n:

$$\backslash\mathrm{pr}\backslash\mathrm{x} = \backslash\mathrm{x}' | \backslash\mathrm{y} = \backslash\mathrm{y}' = \prod_{i=1}^{n} \backslash\mathrm{pr}\backslash\mathrm{x}_i = \backslash\mathrm{x}'_i | \backslash\mathrm{y} = \backslash\mathrm{y}'$$

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

Thus the "naive" assumption has many benefits !

What about computing the *unconditional* $\backslash\mathrm{pr}\backslash\mathrm{x} = \backslash\mathrm{x}'$?

We can obtain this from conditional probabilities as well

$$\backslash\mathrm{pr}\backslash\mathrm{x} = \backslash\mathrm{x}' = \sum_{c \in C} \backslash\mathrm{pr}\backslash\mathrm{x} = \backslash\mathrm{x}' | \backslash\mathrm{y} = c * \backslash\mathrm{pr}\backslash\mathrm{y} = c$$

That is, the unconditional probability follows from the

- Conditional probability given $\backslash\mathrm{y}$
- Weighted by the probability $\backslash\mathrm{pr}\backslash\mathrm{y}$ for each possible value of $\backslash\mathrm{y}$

This follows from the definition of conditional probability.

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

- The unconditional probabilities $\backslash\mathrm{pr}\backslash\mathrm{y}$
    - Depends on number of classes $|C|$
- Probabilities conditional on $\backslash\mathrm{y}$: $\backslash\mathrm{pr}\backslash\mathrm{x}|\backslash\mathrm{y}$
    - Depends on length of $\backslash\mathrm{x}$: n

Example

Here is a hypothetical trading example for equities

- There are two categorical features
    - Valuation: possible values $\{\text{Rich}, \text{Cheap}\}$
        - Is the current stock price expensive (Rich) or inexpensive (Cheap) ?
    - Yield: possible values $\{\text{High}, \text{Low}\}$
        - Is the dividend yield of the stock desirable (High) or undesirable (Low) ?
- Target: An Action with possible values $\{\text{Long}, \text{Short}, \text{Neutral}\}$
    - 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]:
```python
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

Note

- we are displaying only the first 5 examples in each slice (i.e, we use Pandas `.head()` for display)
- so the total number of examples displayed is less than m, the size of the training dataset

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

- row i conditions on $\backslash\mathbf{y} = \backslash\mathbf{y}'$ where $\backslash\mathbf{y}' \in \{\text{Long}, \text{Neutral}, \text{Short}\}$
- each column conditions on one pair $(\backslash\mathbf{x}_1', \backslash\mathbf{x}_2') \in \{\text{Cheap}, \text{Fair}, \text{Rich}\} \times \{\text{High}, \text{Low}\}$

These are counts which can easily be converted into probabilities.

Let's parse this table:

- Columns: $\backslash\mathrm{cnt}\backslash\mathrm{y}|\backslash\mathrm{x}$
    - A column (defined by concrete values for each of the two attributes)
    - Defines a distribution over the target (Action)
- Column Sum: $\backslash\mathrm{cnt}\backslash\mathrm{x} = \sum_{a \in \mathrm{Action}} \backslash\mathrm{cnt}\backslash\mathrm{x}|a$
    - Total number of examples with attribute pair $\backslash\mathrm{x}$
- Rows: $\backslash\mathrm{cnt}\backslash\mathrm{x}|\backslash\mathrm{y}$
    - 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: $\backslash\mathrm{cnt}a = \sum_{\backslash\mathrm{x}} \backslash\mathrm{cnt}\backslash\mathrm{x}|a$
    - 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=["Valuatio
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 $\backslash\mathbf{x}_j$

- 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
$$\text{Is}_{\backslash\mathbf{x}_j < t_j}$$
- Divide the range of values into multiple intervals ("buckets") by a sequence of thresholds defining boundaries
$$\text{Is}_{t_{j,l-1} \le \backslash\mathbf{x}_j \le t_{j,l}}$$

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.

But this "counting" works

- only if all features are *from a finite set of distinct values*

We can't apply counting to continuous-valued features.

Moreover: we can predict a target, given out-of-sample feature vector $\color{red}{\backslash\mathbf{x}}'$

- only if there is at least one *training example* with feature $\color{red}{\backslash\mathbf{x}}_j = \color{red}{\backslash\mathbf{x}}'_j$
  - for each feature j
- else there is *no count* conditional on $x\_j = \color{red}{\backslash\mathbf{x}}'_j$ (the *zero frequency* problem)
  - and conditioning on $\color{red}{\backslash\mathbf{x}}_j = \color{red}{\backslash\mathbf{x}}'_j$ results in *division by 0*

Additive smoothing

Before we abandon counting, we note that

- there are simple (but unsatisfying) solutions
- to the division by zero and other zero frequency related problems

We simply inflate all counts by some parameter α.

This eliminates divide-by-zero issues but biases the counts.

This is called additive smoothing (https://en.wikipedia.org/wiki/Additive_smoothing)

Representing the marginal probability by a function

The typical solution to the counting issues is to represent each feature/target

- by a continuous probability distribution
- defined by a functional form
    - parameterized by empirical counts obtained from the training examples

For example,

- numeric $\backslash\mathbf{x}_1$ can be represented as a
    - Normal distribution with mean $\backslash\bar{\mathbf{x}}_1$ and standard deviation $\sigma(\backslash\mathbf{x}_1)$
- binary categorical $\backslash\mathbf{x}_1$ can be represented as a
    - Bernoulli distribution with probability $\frac{\text{count}(\backslash\mathbf{x}_1 = 1)}{m}$

We note that `sklearn` doesn't even offer the "counting" option

- one must specify empirical distributions (?)

Non-the-less: counting is a great pedagogical tool for motivating Naive Bayes.

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
```