Universidade Presbiteriana Mackenzie

Disciplina: Classificação

Classificação Bayesiana

Professor: Bruno Silva (ibm.biz/brunosilva)

Baysean Classification

- Bayesian reasoning is the formal process we use to update our beliefs about the world once we've observed some data
- Bayesian statistics is closely aligned with how people naturally use evidence to reason about everyday problems
- Tricky part is breaking down this natural thought process into a rigorous, mathematical one

Reasoning About Strange Experiences

Reference: Pixabay https://pixabay.com/service/license/

Reasoning About Strange Experiences

One night you are suddenly awakened by a bright light at your window

You jump up from bed and look out to see a large object in the sky that can only be described as saucer shaped

Could this be a UFO?!

Reference: Pixabay https://pixabay.com/service/license/

Bayesian reasoning

- This reasoning tends to happen **so quickly** that you don't have any time to **analyze your own thinking**
- You created a new belief without questioning it
- You did not believe in the existence of UFOs
- You've updated your beliefs and now think you've seen a UFO

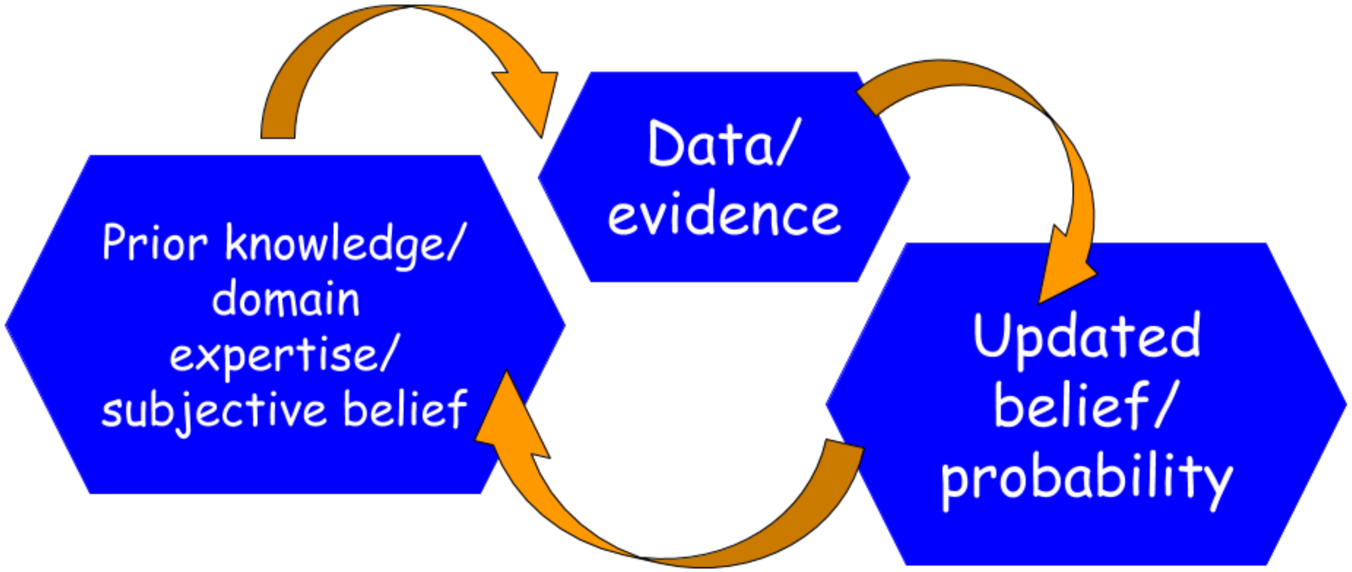
Bayesian reasoning involves

- You’re confronted with a situation
- Making probabilistic assumptions
- Then using those assumptions to update your beliefs about the world

Bayesian reasoning

1. Observed data
2. Formed a hypothesis
3. Updated your beliefs based on the data

Bayesian reasoning

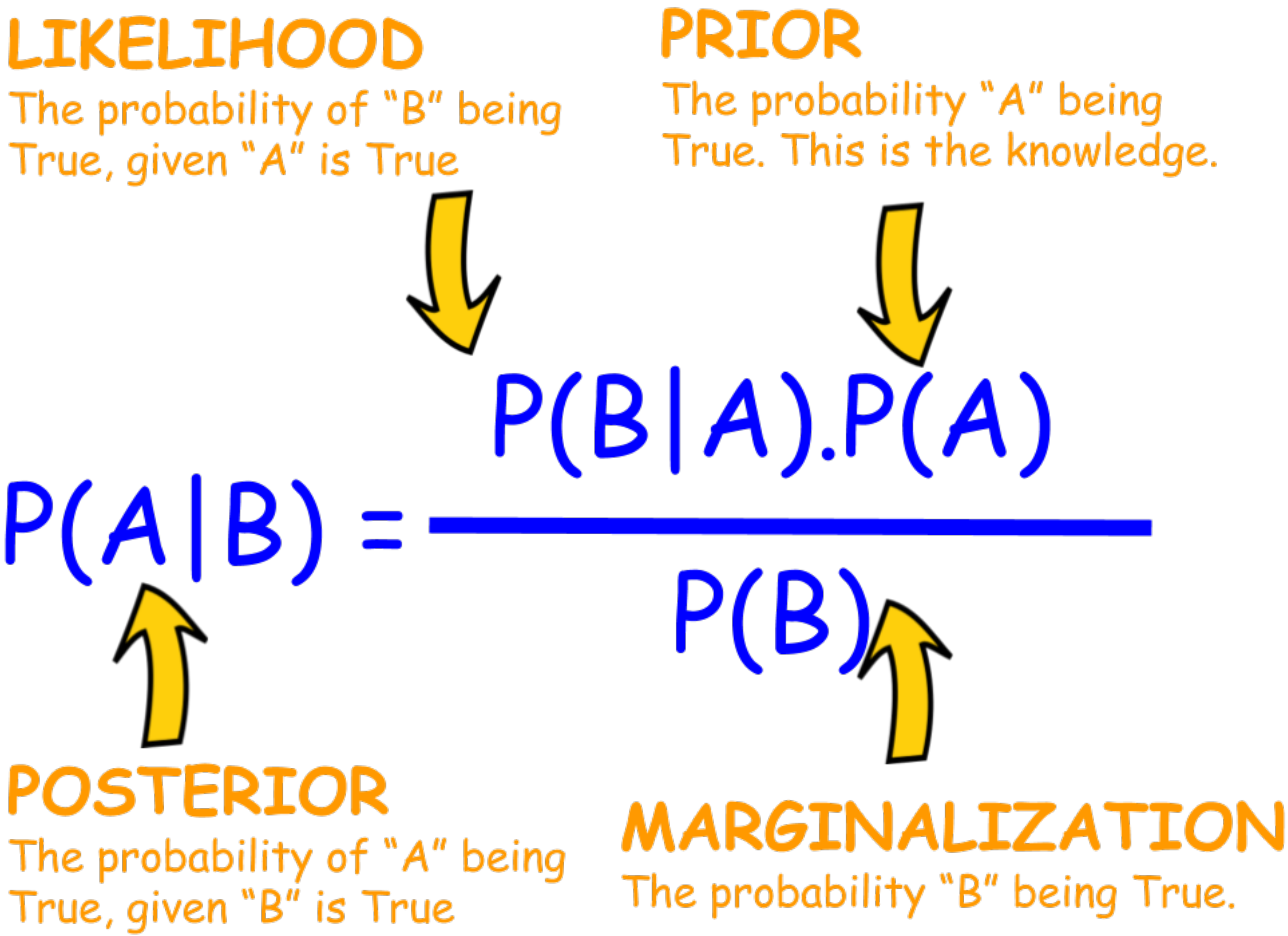


From: https://towardsdatascience.com/bayes-rule-with-a-simple-and-practical-example-2bce3d0f4ad0

Bayes Himself

From: "https://en.wikipedia.org/wiki/Thomas_Bayes"

Bayes Rule



From: https://towardsdatascience.com/bayes-rule-with-a-simple-and-practical-example-2bce3d0f4ad0

It is a powerful law of probability that brings in the concept of ‘subjectivity’ or ‘the degree of belief’ into the cold, hard statistical modeling.

[Reference (https://towardsdatascience.com/bayes-rule-with-a-simple-and-practical-example-2bce3d0f4ad0)](https://towardsdatascience.com/bayes-rule-with-a-simple-and-practical-example-2bce3d0f4ad0)

If the data support the hypothesis then the probability goes up, if it does not match, then probability goes down.

[Reference (https://towardsdatascience.com/bayes-rule-with-a-simple-and-practical-example-2bce3d0f4ad0)](https://towardsdatascience.com/bayes-rule-with-a-simple-and-practical-example-2bce3d0f4ad0)

Bayesian Inference Applications

- Genetics
- Linguistics
- Image processing
- Brain imaging
- Cosmology
- Machine learning
- Epidemiology ...

In [9]:

```python
from wand.image import Image as WImage
img = WImage(filename='https://arxiv.org/pdf/1812.06855.pdf', height=600); img
```

Out[9]:

Bayesian Optimization in AlphaGo

Yutian Chen, Aja Huang, Ziyu Wang, Ioannis Antonoglou, Julian Schrittwieser, David Silver & Nando de Freitas

DeepMind, London, UK
yutianc@google.com

Abstract

During the development of AlphaGo, its many hyper-parameters were tuned with Bayesian optimization multiple times. This automatic tuning process resulted in substantial improvements in playing strength. For example, prior to the match with Lee Sedol, we tuned the latest AlphaGo agent and this improved its win-rate from 50% to 66.5% in self-play games. This tuned version was deployed in the final match. Of course, since we tuned AlphaGo many times during its development cycle, the compounded contribution was even higher than this percentage. It is our hope that this brief case study will be of interest to Go fans, and also provide Bayesian optimization practitioners with some insights and inspiration.

1 Introduction

Bayesian optimization was used as a routine service to adjust the hyper-parameters of AlphaGo [Silver et al., 2016] during its design and development cycle, resulting in progressively stronger agents. In particular, Bayesian optimization was a significant factor in the strength of AlphaGo in the highly publicized match against Lee Sedol.

AlphaGo may be described in terms of two stages: Neural network training, and game playing with Monte Carlo tree search (MCTS). Each of these stages has many hyper-parameters. We focused on tuning the hyper-parameters associated with game playing. We did so because we had reasonably robust strategies for tuning the neural networks, but less human knowledge on how to tune AlphaGo during game playing.

We meta-optimized many components of AlphaGo. Notably, we tuned the MCTS hyper-parameters, including the ones governing the UCT exploration formula, node-expansion thresholds, several hyper-parameters associated with the distributed implementation of MCTS, and the hyper-parameters of the formula for choosing between fast roll-outs and value network evaluation per move. We also tuned the hyper-parameters associated with the evaluation of the policy and value networks, including the softmax annealing temperatures. Finally, we meta-optimized a formula for deciding the search time per move during games. The number of hyper-parameters to tune varied from 3 to 10 depending on a tuning task. The results section of this brief paper will expand on these tasks.

Bayesian optimization not only reduced the time and effort of manual tuning, but also improved the playing strength of AlphaGo by a significant margin. Moreover, it resulted in useful insights on the individual contribution of the various components of AlphaGo, for example shedding light on the value of fast Monte Carlo roll-outs versus value network board evaluation.

There is no analytically tractable formula relating AlphaGo's win-rate and the value of its hyper-parameters. However, we can easily estimate it via self-play, that is by playing an AlphaGo version v against a baseline version v_0 for N games and, subsequently, computing the average win-rate:

Practical Example

Suppose that a test for using a particular drug is 97% sensitive and 95% specific. That is, the test will produce 97% true positive results for drug users and 95% true negative results for non-drug users. Suppose, we also know that 0.5% of the general population are users of the drug.

What is the probability that a randomly selected individual with a positive test is a drug user?

Bayes Rule

- A - Is a drug user, $\bar{A}$ - Not a drug user
- B - Positive test, $\bar{B}$ - Negative Test

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

$$P(B) = P(B|A) \times P(A) + P(B|\bar{A}) \times P(\bar{A})$$

Bayes Rule

LIKELIHOOD
The probability of "B" being True, given "A" is True

PRIOR
The probability "A" being True. This is the knowledge.

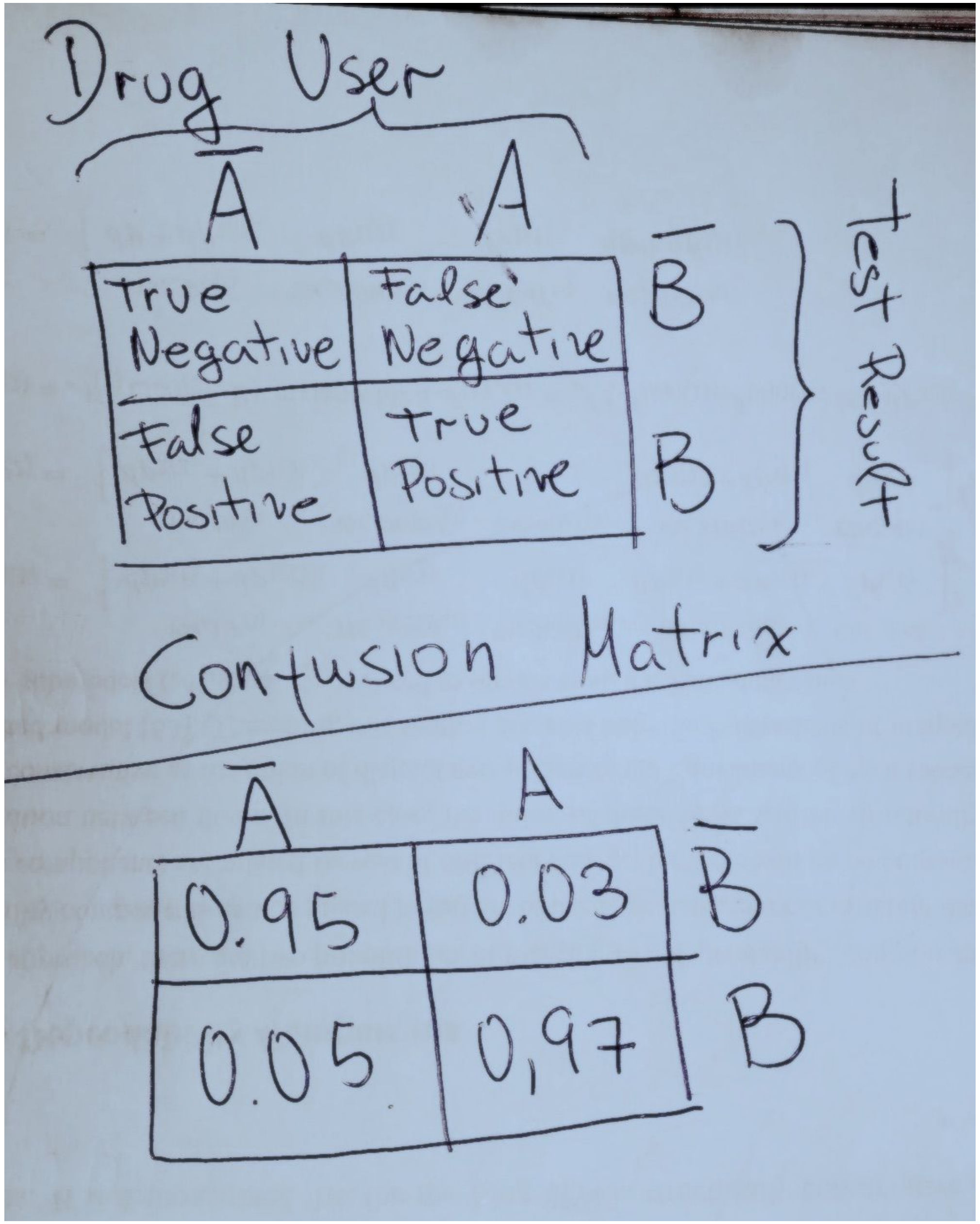
$$P(A|B) = \frac{P(B|A).P(A)}{P(B)}$$

POSTERIOR
The probability of "A" being True, given "B" is True

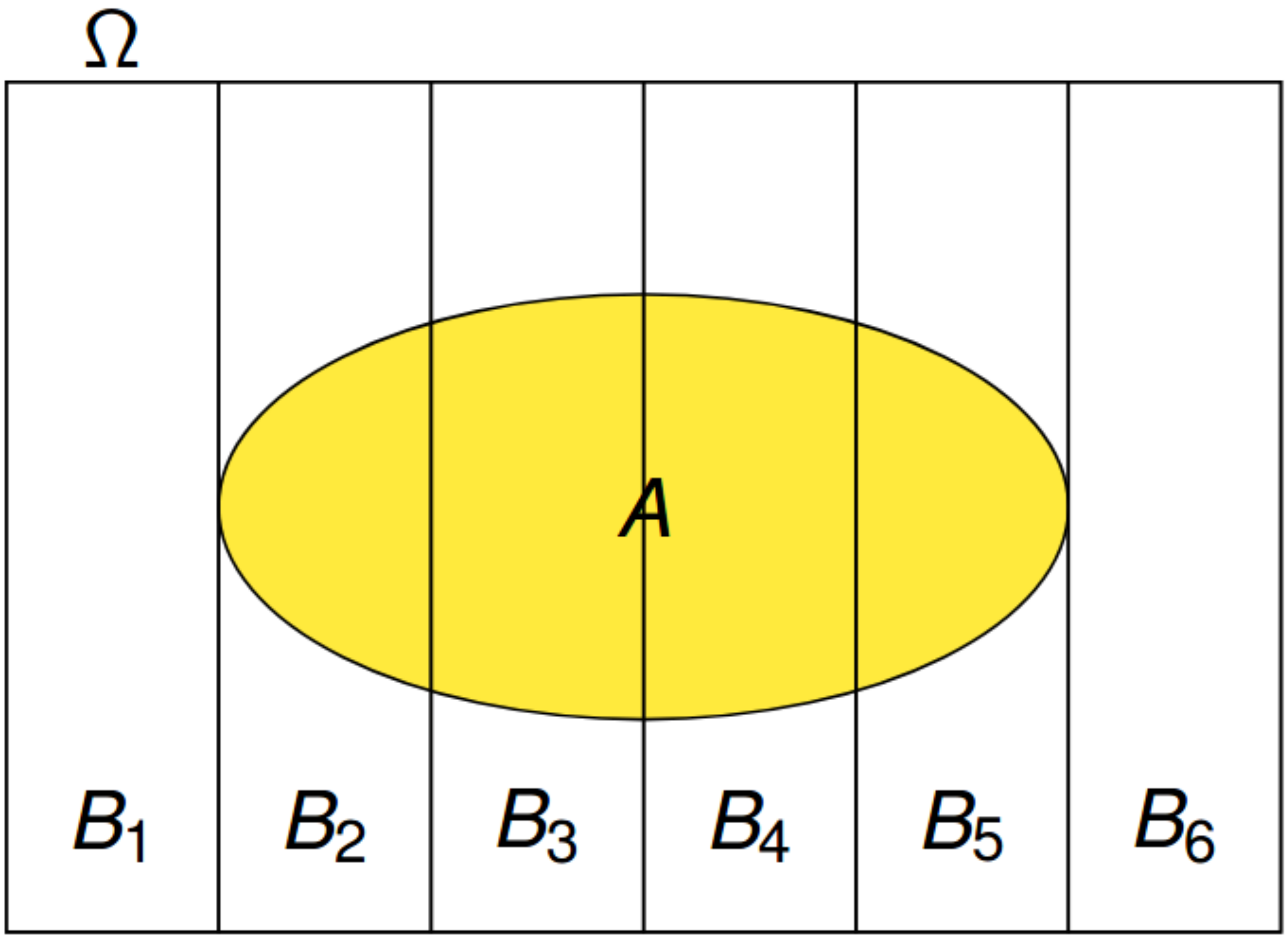
MARGINALIZATION
The probability "B" being True.

From: https://towardsdatascience.com/bayes-rule-with-a-simple-and-practical-example-2bce3d0f4ad0

Bayes Rule


Drug User
Ā
A
True Negative
False Negative
B̄
False Positive
True Positive
B
Test Result
Confusion Matrix
Ā
A
0.95
0.03
B̄
0.05
0,97
B

Law of total probability



From: http://khannay.com/StatsBook/probability.html

Bayes Rule

$$P(B) = P(B|A) \cdot P(A) + P(B|\bar{A}) \cdot P(\bar{A})$$

$$P(B|A) = 0.97 \quad \Big| \quad P(B|\bar{A}) = 0.05$$

$$P(\bar{B}|A) = 0.03 \quad \Big| \quad P(\bar{B}|\bar{A}) = 0.95$$

$$P(B) = 0.97 * 0.005 + 0.05 * 0.995$$

$$\boxed{P(B) = 0.0546}$$

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

$$= \frac{0.97 * 0.005}{0.0546}$$

$$\simeq 0.088$$

Direct Calculation

In [3]:

```
P_A = 0.005
P_BgA = 0.97
P_nA = 1 - P_A
P_BgnA = 0.05

P_B = P_BgA * P_A + P_BgnA * P_nA
P_AgB = (P_BgA * P_A) / P_B
P_AgB
```

Out[3]:

```
0.08882783882783883
```

Fancy function

[https://gist.github.com/tirthajyoti/46453f215156c21d47d34cabd07b5d58 (https://gist.github.com/tirthajyoti/46453f215156c21d47d34cabd07b5d58)](https://gist.github.com/tirthajyoti/46453f215156c21d47d34cabd07b5d58)

In [4]:

```python
def drug_user(sensitivity=0.99, specificity=0.99, prevelance=0.01, verbose=True):
    p_user = prevelance
    p_non_user = 1-prevelance
    p_pos_user = sensitivity
    p_neg_user = specificity
    p_pos_non_user = 1-specificity

    num = p_pos_user*p_user
    den = p_pos_user*p_user+p_pos_non_user*p_non_user

    prob = num/den

    return prob

drug_user(sensitivity=0.97, specificity=0.95, prevelance=0.005)
```

Out[4]:

```
0.08882783882783876
```

Spam Filter Example

- You are a data scientist and wants to build a spam filter
- Imagine a “universe” that consists of receiving a message chosen randomly from all possible messages
- Let S be the event “the message is spam” and B be the event “the message contains the word bitcoin.
- Bayes’s theorem tells us that the probability that the message is spam conditional on containing the word bitcoin is

$$P(S|B) = \frac{P(B|S)P(S)}{P(B|S)P(S) + P(B|\neg S)P(\neg S)}$$

Bayes Rule

- Assume that any message is equally likely to be spam or not spam ($P(S) = P(\neg S) = 0.5$)

$$P(S|B) = \frac{P(B|S)}{P(B|S) + P(B|\neg S)}$$

- 50% of spam messages have the word bitcoin
- 1% of nonspam messages have the word bitcoin
- What is the probability that any given bitcoin-containing email?

$$0.5/(0.5 + 0.01) = 98\%$$

Important Points

- The crucial point here is the prior
- Piece of generalized knowledge about the common prevalence rate
    - 0.5% chance of that person being a drug-user
- With the new evidence (tested positive)
- We update our beliefs and increase the probability of being a drug user

Naive Bayes Classifier

In [299]:

```
import pandas as pd
dataset = pd.read_csv("https://bit.ly/39EzSm5")
dataset
```

Out[299]:

	outlook	temp	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes
5	rainy	cool	normal	True	no
6	overcast	cool	normal	True	yes
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
9	rainy	mild	normal	False	yes
10	sunny	mild	normal	True	yes
11	overcast	mild	high	True	yes
12	overcast	hot	normal	False	yes
13	rainy	mild	high	True	no

Naive Bayes

- The presence of one particular feature does not affect the other. Hence it is called naive.
- We assume that if it is raining, it is not necessarily cold ...

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

- X is the set of features $X = (x_1, x_2, x_3, \ldots, x_n)$
- y is the label

Naive Bayes

- Label probability is the following
$$P\left(y|x_1,\ldots,x_n\right) = \frac{P\left(x_1|y\right)P\left(x_2|y\right)\ldots P\left(x_n|y\right)P(y)}{P\left(x_1,x_2,\ldots,x_n\right)}$$

- Now, you can obtain the values for each $P(x_z|y)$ by looking at the dataset and substitute them into the equation.

- $P\left(x_1\right)P\left(x_2\right)\ldots P\left(x_n\right)$ is constant
$$P\left(y|x_1,\ldots,x_n\right) \propto P(y)\prod_{i=1}^{n}P\left(x_i|y\right)$$

Naive Bayes

y will be the class that maximizes the previous equation

$$y = \operatorname{argmax}_y P(y) \prod_{i=1}^{n} P\left(x_i \mid y\right)$$

Example from scratch

Should we play if conditions are: [rainy, cool, humidity_normal, not_windy] ?

In [301]:

```
values = dataset.play.value_counts() / len(dataset.play)
prob_labels = values.to_dict()
prob_labels
```

Out[301]:

```
{'yes': 0.6428571428571429, 'no': 0.35714285714285715}
```

In [245]:
```
# outlook for yes
dataset_filtered = dataset[dataset.play == "yes"]
size = len(dataset_filtered)
values = dataset_filtered['outlook'].value_counts() / size
outlook_yes_proportions = values.to_dict()
outlook_yes_proportions
```

Out[245]:
```
{'overcast': 0.4444444444444444,
 'rainy': 0.3333333333333333,
 'sunny': 0.2222222222222222}
```

In [ ]:
```
```

In [304]:

```
def get_proportions(feature, given):
    # outlook for yes
    dataset_filtered = dataset[dataset.play == given]
    size = len(dataset_filtered)
    values = dataset_filtered[feature].value_counts() / size
    outlook_yes_proportions = values.to_dict()
    return outlook_yes_proportions
```

In [308]:

```
get_proportions('temp', 'no')
```

Out[308]:

```
{'hot': 0.4, 'mild': 0.4, 'cool': 0.2}
```

In [257]:

```
outlook_yes_proportions = get_proportions('outlook', 'yes')
temp_yes_proportions = get_proportions('temp', 'yes')
humidity_yes_proportions = get_proportions('humidity', 'yes')
windy_yes_proportions = get_proportions('windy', 'yes')
```

In [310]:

```
humidity_yes_proportions
```

Out[310]:

```
{'normal': 0.6666666666666666, 'high': 0.3333333333333333}
```

In [256]:

```python
outlook_no_proportions = get_proportions('outlook', 'no')
temp_no_proportions = get_proportions('temp', 'no')
humidity_no_proportions = get_proportions('humidity', 'no')
windy_no_proportions = get_proportions('windy', 'no')
```

In [313]:

```python
windy_no_proportions
```

Out[313]:

```
{True: 0.6, False: 0.4}
```

For each label evaluate probabilities

Should we play if conditions are: [rainy, cool, humidity_normal, not_windy] ?

$$P\left(y|x_1, \ldots, x_n\right) = \frac{P\left(x_1|y\right) P\left(x_2|y\right) \ldots P\left(x_n|y\right) P(y)}{P\left(x_1, x_2, \ldots, x_n\right)}$$

In [325]:

```
# Should we play if conditions are: [rainy, cool, humidity_normal, not_windy] ?
Pn_yes = (
    outlook_yes_proportions["sunny"]*
    temp_yes_proportions['cool']*
    humidity_yes_proportions['normal']*
    windy_yes_proportions[True]
)*prob_labels['yes']

Pn_yes
```

Out[325]:

```
0.010582010582010581
```

In [326]:

```python
# Should we play if conditions are: [rainy, cool, humidity_normal, not_windy] ?
Pn_no = (
    outlook_no_proportions["sunny"]*
    temp_no_proportions['cool']*
    humidity_no_proportions['normal']*
    windy_no_proportions[True]
)*prob_labels['no']

Pn_no
```

Out[326]:

```
0.005142857142857143
```

Probability allways sum to 1

$$P(y = yes|X) + P(y = no|X) = 1$$

In [327]:

```
P_yes = Pn_yes / (Pn_yes + Pn_no)
P_yes
```

Out[327]:

```
0.6729475100942126
```

In [330]:

```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import BernoulliNB
from sklearn import preprocessing

#Create a Gaussian Classifier
model = BernoulliNB()


le = preprocessing.OrdinalEncoder()
ly = preprocessing.LabelEncoder()

X = le.fit_transform(dataset.iloc[:, :-1])
y = ly.fit_transform(dataset.iloc[:, -1])

# Train the model using the training sets
model.fit(X,y)
matrix = le.transform([["sunny", "cool", "normal", True]])
model.predict_proba(matrix)
```

Out[330]:

```
array([[0.32612666, 0.67387334]])
```

Example with `scikit-learn`

In [186]:

```
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']
```

Encoding Features

In [187]:

```
from sklearn import preprocessing


le=preprocessing.LabelEncoder()
weather_encoded=le.fit_transform(weather)
print(weather_encoded)
```

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

Encode temp and play columns

In [49]:

```
temp_encoded=le.fit_transform(temp)

label=le.fit_transform(play)

print(temp_encoded, label)
```

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

Combinig weather and temp into single list of tuples

In [50]:

```
features=list(zip(weather_encoded,temp_encoded))
print(features)
```

```
[(2, 1), (2, 1), (0, 1), (1, 2), (1, 0), (1, 0), (0, 0), (2, 2), (2, 0), (1,
2), (2, 2), (0, 2), (0, 1), (1, 2)]
```

In [51]:

```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import MultinomialNB

#Create a Gaussian Classifier
model = MultinomialNB()

# Train the model using the training sets
model.fit(features,label)

predicted= model.predict([[2,2]]) # 0:Overcast, 2:Mild
print(f"Predicted Value: {predicted}" )
```

```
Predicted Value: [1]
```

References and Further Reading

1. Bayesian Statistics the Fun Way by Will Kurt Published by No Starch Press, 2019
2. Data Science from Scratch, 2nd Edition by Joel Grus Published by O'Reilly Media, Inc., 2019
3. Bayes' rule with a simple and practical example: https://towardsdatascience.com/bayes-rule-with-a-simple-and-practical-example-2bce3d0f4ad0 (https://towardsdatascience.com/bayes-rule-with-a-simple-and-practical-example-2bce3d0f4ad0)
4. Original Bayes article: https://royalsocietypublishing.org/doi/pdf/10.1098/rstl.1763.0053 (https://royalsocietypublishing.org/doi/pdf/10.1098/rstl.1763.0053)
5. Bayesian Optimization in AlphaGo: https://arxiv.org/abs/1812.06855 (https://arxiv.org/abs/1812.06855)
6. Probability Theory: https://towardsdatascience.com/probability-theory-continued-infusing-law-of-total-probability-4abfca6e65bb (https://towardsdatascience.com/probability-theory-continued-infusing-law-of-total-probability-4abfca6e65bb)
7. Naive Bayes Classifier: https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c (https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c)