Naive Bayes and Decision Trees

Alipio Jorge (DCC-FCUP)

November 2020

Classification

The credit office of the bank now has to decide whether to give the loan or not to a specific client. Management feels that the current credit decision procedure is not efficient and that the bank can make more money and secure more good clients with a better process The bank has an historical record of loans. Some were conceded and went well. Other were not conceded or had a bad outcome.

- What is the business problem?
- The machine learning problem?
- What kind of **task** is it?

Classification: types

- Decide if a new case belongs to a set of known classes
- a type of **supervised learning**
- **Binary classification**
  - Bank credit
  - Has cancer
  - This email is spam
  - Is the market going up or down tomorrow?
- **Multiclass**
  - Which disease does the patient have
  - Which folder for this email
  - What is the type of galaxy in this image

Classification: process

- The Process (simplified CRISP)
  - Define problem
  - Prepare data
  - Build model (**classifier**) by applying a **learning algorithm**
  - Evaluate model
  - Deploy

Classification: setup

- The data (most common setup)
  - **Examples** are pairs $< x, y >$, also called **tuples**
    - $< x, y > \in D$ where D is the **dataset**
    - x_i is one row of of a $n \times m$ table X
    - The dimensions / columns of X are the **attributes**
    - y_i is the **class** of x_i
    - $y_i \in Y = Classes = \{C_1, C_2, \ldots, C_k\}$
    - y_i also called **labels**

Classification: aim

- The **aim** of classification
  - **Given** a dataset D of pairs $< x, y >$
  - **Obtain**
    - a function $\hat{f} : X \rightarrow Classes$
    - such that $\hat{f}$ **approximates** an unknown function $f(x)$ that assigns labels to objects

Classification: the Bayesian view

- How can $\hat{f}(x)$ be found?
- Suppose
  - we have a **new case** x to classify
- We want the class C_{max} that maximises $P(C_j \mid x)$
  - so, *all* we have to do is **estimate** $P(C_j \mid x)$ for each class
- How?
  - lots of different ways
- A simple and **principled** one?
  - **Naive Bayes**

Naive Bayes

- Bayes theorem:

$$P(C_j \mid x) = \frac{P(x \mid C_j).P(C_j)}{P(x)}$$

- The class C_{max} that maximises this is the **maximum posteriori hypothesis**
- How is this calculated?
  - we assume that $P(x)$ is **constant**
  - we estimate $P(C_j) = \frac{|x_i \in C_j|}{|D|}$ from the data
  - and estimate $P(x \mid C_j)$ – this is **trickier**

The Naive Bayes trick

- In general estimating $P(x \mid C_j)$ is **hard**
  - data is **sparse**
  - approximations can be **computationally expensive**
- The **trick** is ti naïvely assume
  - the attributes are **class-conditional independent**

$$P(A_i \mid C, A_j) = P(A_i \mid C)$$

The Naive Bayes trick

- This assumption is **not realistic**
  - but it is **good enough** to make the approach useful
- This assumption greatly **simplifies** the computation

$$P(x \mid C_j) = \prod_{i=1}^{n} P(x_i \mid C_j)$$

- Now, each $P(x_i \mid C_j)$ is easy to estimate - Besides, it is **well founded** - driven from **first principles** - not **ad hoc**

How Naive Bayes works

- Estimate the probabilities of $P(x_{i,A}|C)$ from example i and attribute A
  - A is **categorical**:
    - the number of tuples of class C with value $x_{i,A}$ in A
    - divided by the size of class C
  - A is **continuous**:
    - assume a **Gaussian distribution**
    - estimate mean and standard deviation from sample of each A for each class

Naive Bayes: example

- The `german_credit_data.csv` in kaggle (adapted from UCI)
- predict **Risk** (classes good and bad)

```
import pandas as pd
d=pd.read_csv('../Dados/german_credit_data.csv')
d[['Age','Sex','Risk']]
```

```
     Age     Sex  Risk
0     67    male  good
1     22  female   bad
2     49    male  good
3     45    male  good
4     53    male   bad
5     35    male  good
6     53    male  good
7     35    male  good
8     61    male  good
```

Naive Bayes: example

- $P(Sex = male \mid Risk = good)$

```
freq_male_good=len(d[(d['Sex']=='male') & (d['Risk']=='good')]
freq_good=len(d[d['Risk']=='good'])
prob_male_good=freq_male_good/freq_good
prob_male_good
```

```
0.7128571428571429
```

Naive Bayes: example

- $P(Age = 22 \mid Risk = good)$

```
import numpy as np

mean_age_good=np.mean(d.loc[d['Risk']=='good',['Age']])
std_age_good=np.std(d.loc[d['Risk']=='good',['Age']])

from scipy.stats import norm

prob_22_good=norm.pdf((22-mean_age_good)/std_age_good)
prob_22_good
```

```
array([ 0.18248811])
```

Naive Bayes: example

- $P(Age = 22 \mid Risk = good) \times P(Sex = male \mid Risk = good)$
  - which is proportional to $P(Risk = good \mid x)$

```
prob_male_good*prob_22_good
```

```
array([ 0.13008795])
```

- **Exercise**: write a python program that classifies any new case of this problem using Naive Bayes (without a predefined NB learner)

Naive Bayes: implementations

- SciKitLearn
  - `GaussianNB` for continuous predictors
  - `CategoricalNB`for categorical ones
  - and others...
- Mixed variables
  - `MixedNB` from `mixed-naive-bayes` library
    - not direct
  - other not so straight solutions
- One can always
  - Discretize continuous

Does Naive Bayes have good results?

- **If assumptions hold** NB is **optimal in theory**
  - If we have **enough data** for good estimations
- **If not**
  - it can have **comparatively good results** in some domains
  - useful in **high dimensional** domains
  - there are methods that can **easily beat NB**

Naive Bayes: further notes

- NB has little hyperparameters
- What if one of the probabilities iz **zero**?
  - we can smooth probabilities using the correction of Laplace

$$P_{Laplace}(E) = \frac{E + \lambda}{N + k.\lambda}$$

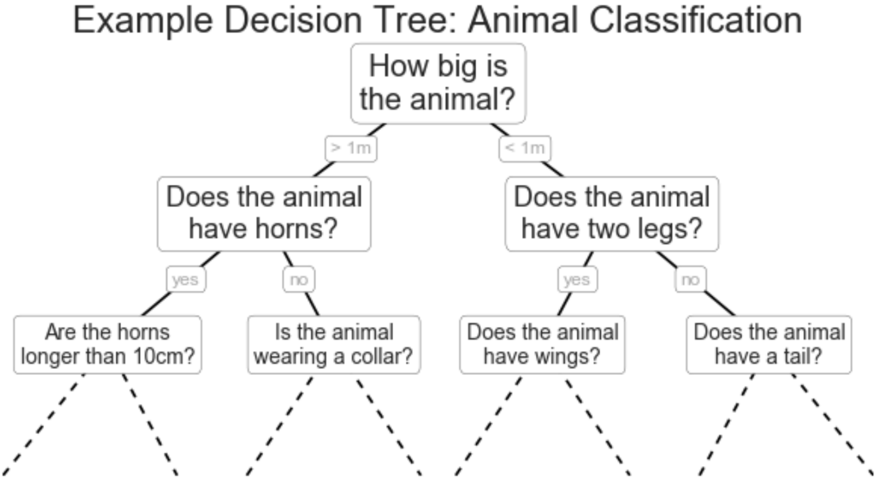
- λ typically is 1, and k is the number of values of E
  - it is like always having one artificial observation per category
- A simple version of NB is easy to implement
  - see https://towardsdatascience.com/introduction-to-na%C3%AFve-bayes-classifier-fa59e3e24aaf

Classification: Decision Trees

- **Where we are**
  - We have seen how to approach **regression** problems using linear regression and the k-Nearest Neighbours (kNN) approach
  - We have seen how to approach **classification** using kNN and naive Bayes
- **Next**
  - we will study **Decision Trees**
    - versatile method
    - handles classification but can be adapted to regression
    - deals with different types of data
    - use a **search** procedure

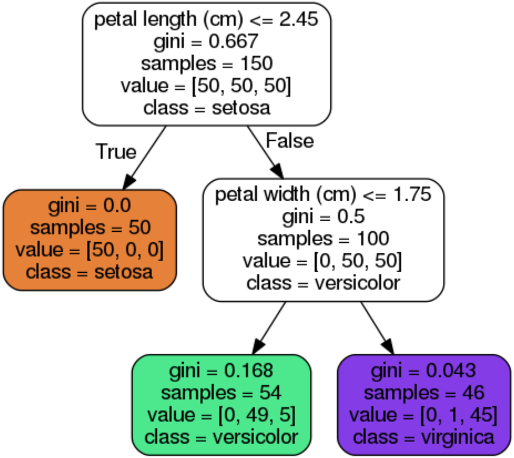
Decision Trees

- The classification function $\hat{f}$ can be implemented as a decision tree

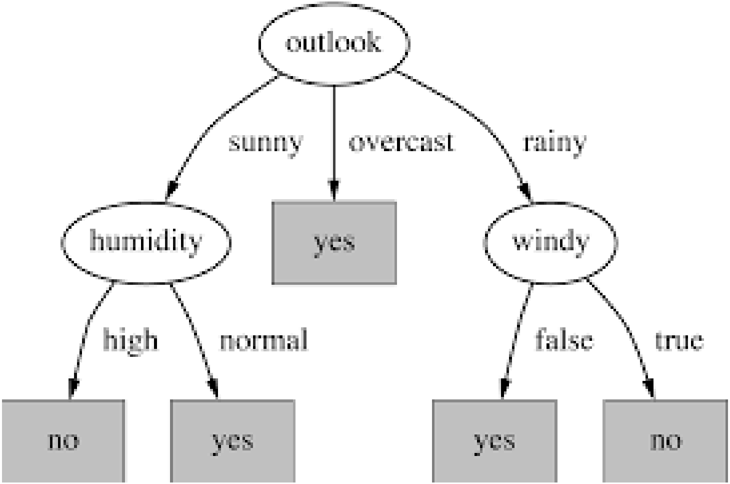


Decision Trees

- We can obtain one for the iris classification problem



Decision Trees: the golf example



How are DT obtained?

- The **idea** is
  - start with all the examples
  - try to **divide** them in two or more groups where classes are as separated as possible
  - **repeat the process** recursively for each group
  - **until** all classes are separated in **small groups**
    - or the groups are too small
- This is called **TDIDT**: Top Down Induction of Decistion Trees

Let's try to do this with the golf data

```python
golf=pd.read_csv('../Dados/golf_df.csv')
golf
```

```
     Outlook Temperature 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
```

The golf data

- Looking at all the examples we have
  - 5 of class **no** and 9 of class **yes**
  - this is already a decision tree (with root only)
  - the majority class wins
- The separation of the classe is not very good, though

The golf data

- Consider now splitting the group according to **Outlook**
  - Outlook has three values
    - **sunny**: (3 no,2 yes)
    - **rainy**: (2,3)
    - **overcast**: (0,4)
  - this gives one **pure** group
    - better than before
    - if `overcast` we go play

The golf data

- Is `Outlook` the best attribute for **splitting**?
    - we have to check with all the attributes
- for now, we will believe it is

The golf data

- Now we try to refine each of the not pure **nodes of the tree**
  - Outlook=sunny
    - Humidity=normal: (0,2)
    - Humidity=high: (3,0)
  - Outlook=rainy
    - Windy=True: (2,0)
    - Windy=False: (0,3)
  - And we end up with a tree that completely separates the classes
    - all **leaves** are pure (one class only)

The golf data

- From the tree we can obtain the **rules**
  - IF Outlook = overcast THEN Play=yes
  - IF Outlook = sunny AND Hum = normal THEN Play=yes
  - IF Outlook = sunny AND Hum = high THEN Play=no
  - IF Outlook = rainy AND Windy = True THEN Play=no
  - IF Outlook = rainy AND Windy = False THEN Play=yes
  - Each rule corresponds to a **branch** of the tree

The golf data

- In the **real world**
  - leaves are not pure
  - datasets have continuous attributes as well
- However, Decision Trees
  - can be useful with real data
  - are human **readable**
  - can be combined in **ensembles** with success

Top Down Induction of Decision Trees

- Selecting the best **split** for each node
- Given attribute A with values $v_1, \ldots, v_m$
  - Calculate **Information Gain** of A
  - measure inspired in information theory

TDIDT: selecting attributes

- How many bits do we need to represent the classes of the examples in a data set D?
  - we have k classes
  - if they are uniformly distributes, *Entropy* is the highest
  - if all the cases are on one class, *Entropy* is **zero**

$$Entropy(D) = -\sum_{i=1}^{k} p_i \log_2(p_i)$$

- Suppose we have 4 classes
  - then we need **two bits** to transmit the label of each example
  - $-0.25 \times \log_2(0.25) = 0.5$
  - $0.5 \times 4 = 2$

TDIDT: selecting attributes

- Now we **choose** the attribute that minimizes *Entropy*
- How do we measure it after a split?
  - we sum the entropies of each resulting group

$$Entropy_A(D) = \sum_{j=1}^{m} \frac{|D_j|}{|D|} \times Entropy(D_j)$$

TDIDT: selecting attributes

- The best attribute is the one that maximizes the **Information Gain**

$$Gain(A) = Entropy(D) - Entropy_A(D)$$

TDIDT: Gain Ratio and the Gini Index

- Information Gain **favours attributes** with many values
  - Dividing in more groups leads more easily to **pure groups**
  - the solution is to use the **Gain Ratio** instead
  - this ratio "discounts" the existence of many values
- Another measure of Attribute split quality
  - the **Gini Index**
  - can be used instead of the information gain

Continuous attributes

- A categorial attribute has natural splits
  - Overcast=sunny, Overcast=rainy, Overcast=overcast
- What are the splits of a **continuous attribute** ?
  - Age<30, Age<44

Age	Risk
23	good
25	good
35	bad
43	bad
45	good
49	good

Tree Pruning

- Trees can easily **overfit**
  - imagine a tree with one example in each leave
- It is often a good idea to **prune** the tree
  - cut some extremities of the branches
- **Prepruning**
  - stop growing the tree
  - if the **complexity** of the tree is too high
- **Post pruning**
  - grow the tree and then cut

Other common parameters

- **Tree depth**
  - controls overfitting but is uninformed
- **Min Split**: Minimum number of cases to have a split
  - controls overfitting
  - we need enough cases for estimating entropy
- **Min Bucket**: Minimum number of cases in a node
  - controls overfitting
  - we need enough cases to decide for a class

Decision tree induction complexity

- If we have N data points what is the **computational complexity** of computing a split for a regression problem?
  - $O(N)$
- And for multilevel trees?
  - tree has maximum depth d : $O(N \cdot d)$
  - assuming the splits tend to be in the middle : $O(N \cdot log_2(N))$
  - in the case of unbalanced splits : $O(N^2)$
- What is the computational complexity if we have p variables?
  - $O(p \cdot N \cdot log_2(N))$
- But remember we have to **sort** the data

Decision trees and search

- The best tree is **not found analytically**
- We **search** for the tree
  - We start with the root (first node)
  - We choose the best split
  - We commit to that choice
  - Until we find a satisfactory solution
- This is a **greedy search** procedure
  - we may find a **local optimum**
  - but it is **efficient**

References

- Books
  - Han, Kamber & Pei, Data Mining Concepts and Techniques, Morgan Kaufman.
- Data
  - https://www.kaggle.com/uciml/german-credit
  - https://www.kaggle.com/priy998/golf-play-dataset
- Blog articles -https://towardsdatascience.com/introduction-to-na%C3%AFve-bayes-classifier-fa59e3e24aaf