FACULTY OF **ENGINEERING AND ARCHITECTURE**

Design of a diagnosis and follow-up platform for patients with chronic headaches

Kiani Lannoye & Gilles Vandewiele

Faculty of Engineering and Architecture

Introduction

Headaches

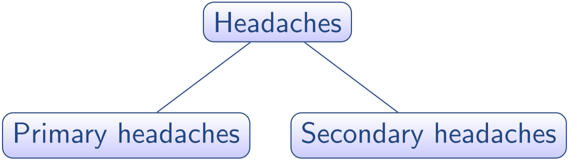
Headaches

Headaches

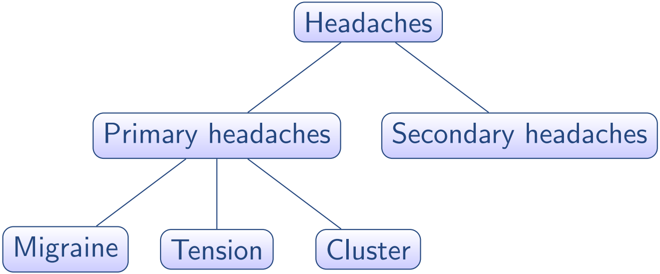
Headaches

Primary headaches

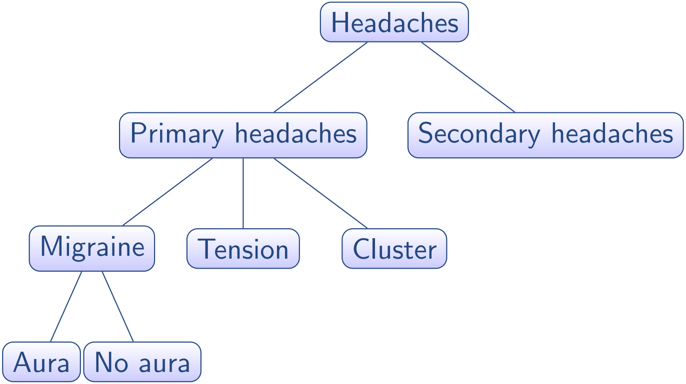
Headaches



Headaches


Headaches
Primary headaches
Secondary headaches
Migraine
Tension
Cluster

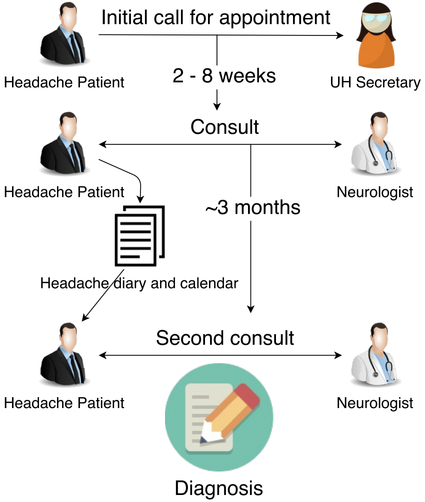
Headaches


Headaches
Primary headaches
Secondary headaches
Migraine
Tension
Cluster
Aura
No aura

Current process UH Ghent

Current process at UH Ghent is:

- Not digital
- cumbersome
- long-lasting


Initial call for appointment
Headache Patient
2 - 8 weeks
UH Secretary
Consult
Headache Patient
Neurologist
~3 months
Headache diary and calendar
Second consult
Headache Patient
Neurologist
Diagnosis

So there is need for a better (digital) alternative! This alternative has to:

- have same capabilities as current solution
- be more efficient
- have more functionality

Platform requirements

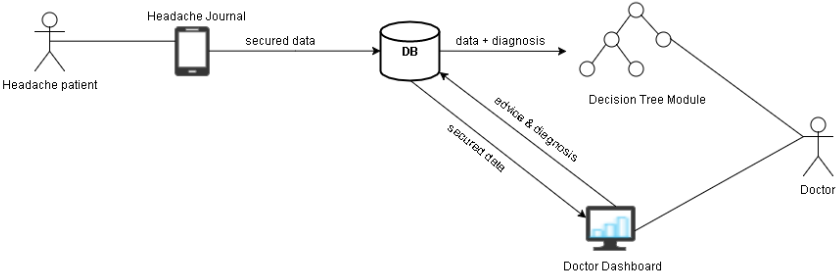
Our proposed alternative consists of:

- Headache journal: mobile app
- Doctor Dashboard: web application
- Machine learning module: auto-classify

Solution non-functional requirements:

- Security
- Availability
- Performance
- Usability

Platform requirements



Mobile Application

Why create a new application?

Competition

- Migraine Buddy
- Headache Diary
- Pfizer headache journal

Mobile Application

Why create a new application?

Competition

- Migraine Buddy
- Headache Diary
- Pfizer headache journal

All good, but:

Mobile Application

Why create a new application?

Competition

- Migraine Buddy
- Headache Diary
- Pfizer headache journal

All good, but:

- none offers usable data export
- none captures all data needed

Development paradigms

Different kinds of approaches for mobile application development:

- Web application
- Hybrid application
- Native application

→ How do we choose?

Web application

Webapps are developed once and can be viewed on (almost) all smartphones (via built-in web engine).

+ "write once, run everywhere" $\Rightarrow$ lower cost
+ No installation required
− limited use of device specific features (GPS, camera, ...)
− **Not all devices same web engines $\Rightarrow$ other view**
− No native look and feel

$\Rightarrow$ No web application

Native application

Native apps are developed once for each OS and installed on the device.

+ Best performance (optimized machine code at compile time)
+ Device specific features usable (GPS, camera, ...)
+ Native look and feel

− **Write code for each OS (very costly dev + maintenance)**
− Installation required

⇒ Native application?

Hybrid application

Hybrid apps are developed once and installed on the device. It uses the devices internal web engine, but has more control than web applications.

+ "write once, run everywhere" ⇒ lower cost
+ Better performance (semi-optimized machine code)
+ Device specific features usable (GPS, camera, ...)
+ Native look and feel (using libraries)
− Installation required
− **Not all devices same web engines ⇒ other view (but manageable)**

⇒ Hybrid application?

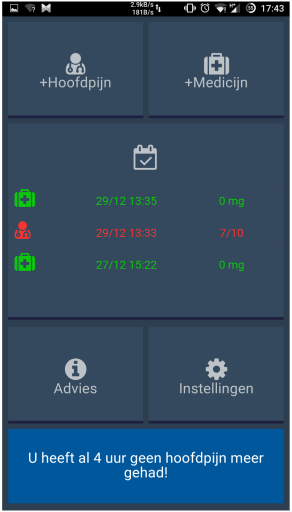
Hybrid vs Native

	Native	**Cross-platform**
+	+ Native UX + device-specific features + Better performance	+ 1 language + Write once, run everywhere + Less maintenance
-	- Multiple languages - Time consuming (development)	- Slower (lower performance) - Less device specific features - Harder to release online (Play Store/App Store)

Hybrid vs Native

	Native	Cross-platform
+	+ Native UX + device-specific features + Better performance	+ 1 language + Write once, run everywhere + Less maintenance
-	- Multiple languages - Time consuming (development)	- Slower (lower performance) - Less device specific features - Harder to release online (Play Store/App Store)

HTML
CSS
Onsen UI
PhoneGap



Chronicals

Intro

Platform requirements

Mobile application

Backend and data exposure

Machine learning - DT's

Genetic merging of DT's

Visualization

Conclusion & future work

Many different induction algorithms



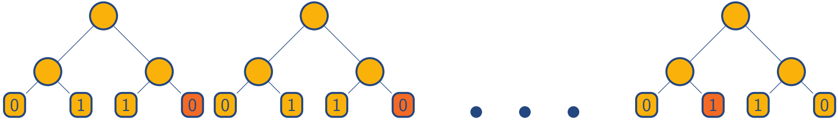
...

C4.5 (C5.0) CART QUEST

→ **Which tree is the most beautiful?**

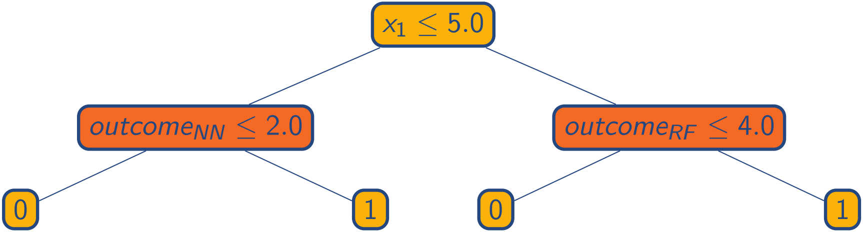
Current ensembles lack interpretability

Boosting, bagging, random forests, etc. require majority voting (classification) or mean calculation (regression) to obtain prediction



Current ensembles lack interpretability

The final decision tree obtained by **stacking** contains uninterpretable internal nodes

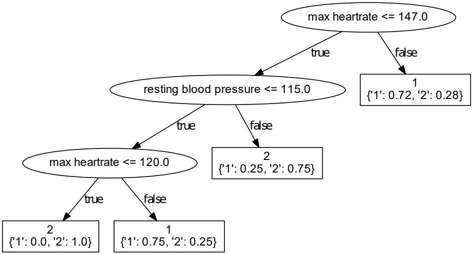


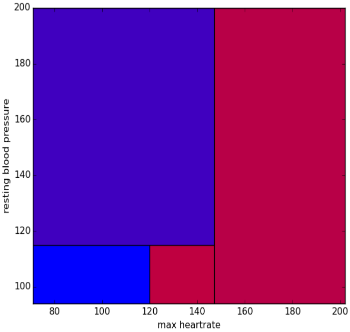
Decision tree $\rightarrow$ decision space

Converting decision trees to decision spaces

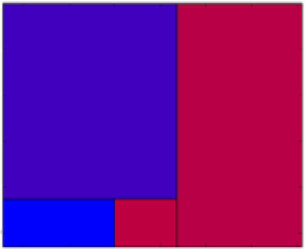
We can define a one-to-one mapping between a decision tree and a set of k-dimensional hyperplanes ($k = \#\text{features}$), called **decision space**. Each node in the decision tree corresponds to a hyperplane in the decision space.

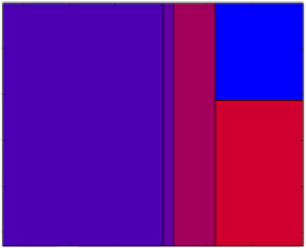
Decision tree → decision space



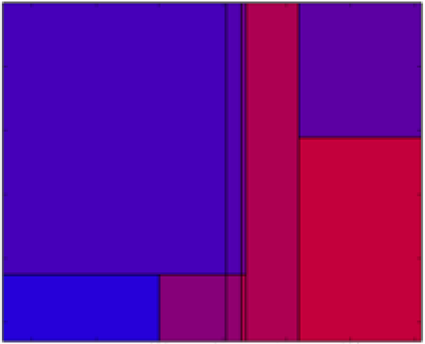


Merging decision spaces

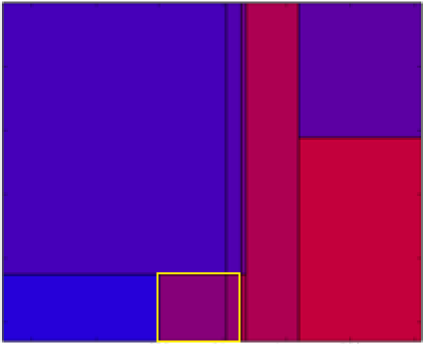




Merging decision spaces



Pruning decision spaces



Decision space $\rightarrow$ decision tree

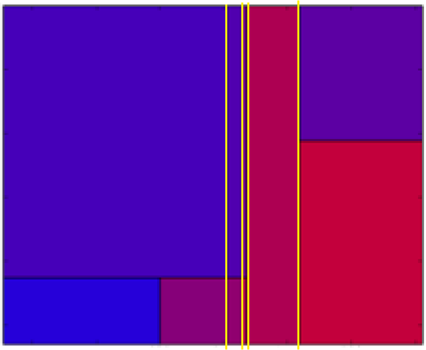
Converting decision spaces to decision trees

One-to-one mapping from decision tree to space is lost during conversion because the order is lost. Therefore, a heuristic approach must be taken, identifying hyperplane candidates and calculating a metric to choose the 'best' plane.

Candidate hyperplanes

In order for a plane to be the next candidate node, it must be unbounded in all dimensions but one.

Decision space → decision tree



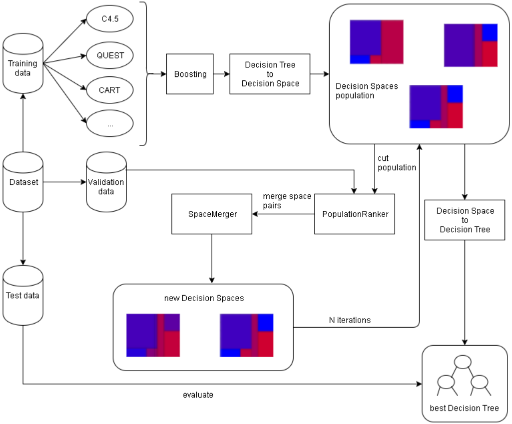
Decision space $\rightarrow$ decision tree

Finding 'best' candidate hyperplane

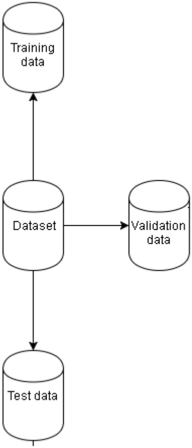
Apply metric function to each plane, these include:

- information gain and Gini from C4.5 and CART respectively
- pick plane from most correlated feature (χ^2- and ANOVA F-test from QUEST)
- pick plane that divide space in two most equal subspaces (using surface/volume or counting number of planes)
- combination

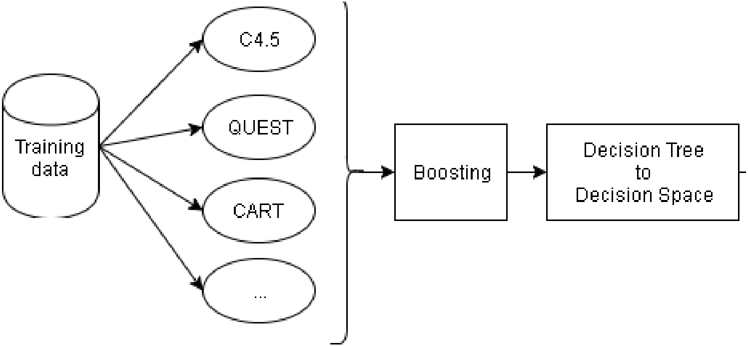
Genetic algorithm



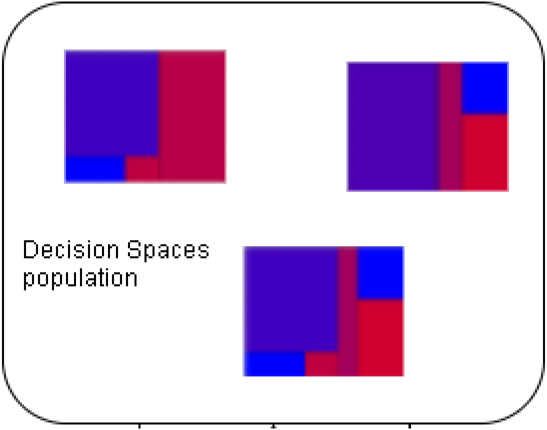
Splitting the data



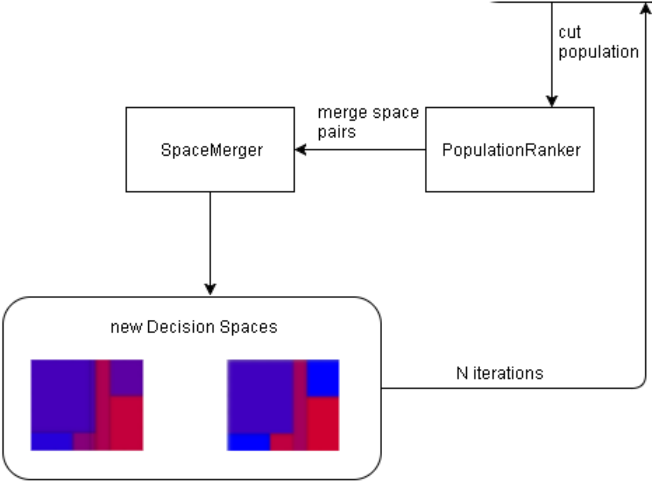
Generate different decision trees



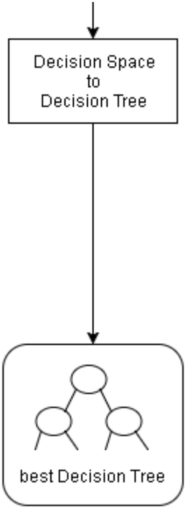
Generate different decision trees



Genetic merging



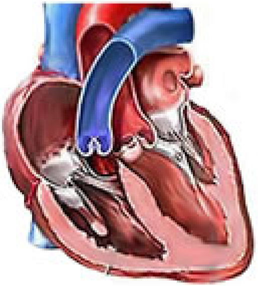
Final iteration



Evaluating our algorithm

To test how good our algorithm performs, we downloaded 5 datasets from the UCI machine learning repository. We then compared our genetic merging algorithm with the three discussed decision tree induction algorithms, using optimal parameters, feature selection when needed and k-fold cross-validation.

Heart disease dataset



- medical dataset
- 270 samples
- 13 features: 6 continuous (max heartrate), 7 discrete (sex)
- 2 classes: sick or healthy
- more healthy than sick samples
- false negatives can cost lives!

Car evaluation dataset

Iris flowers dataset

Shuttle classification dataset

Nursery application dataset

Dataset	Folds	C4.5	CART	QUEST	Genetic
Heart disease	5	**0.8067**	0.7844	0.7844	**0.8067**
	10	**0.8104**	0.7732	0.7881	0.7993
Iris	3	0.9533	0.9467	0.9467	**0.96**
	5	0.9467	0.9333	0.9467	**0.9533**
Cars	3	**0.9722**	0.9693	0.9229	0.9693
	5	0.9711	0.9682	0.9241	**0.9786**
	10	0.9756	0.9751	0.9265	**0.9803**
Shuttle	3	0.9987	0.9983	0.9964	**0.9988**
	5	0.9986	0.9981	0.9962	**0.9988**
	10	0.9990	0.9987	0.9941	**0.9992**
Nursery	3	0.9890	0.9431	0.9147	**0.9914**
	5	0.9918	0.9498	0.9251	**0.9958**
	10	0.9937	0.9568	0.9259	**0.9954**

Bedankt

Bedank voor uw aandacht

No written word,
No spoken plea,
Can teach the youth what they should be,
Nor all the books on all the shelves,
It's what the teachers are themselves