Project Report
Fundamentals of Artificial Intelligence and Knowledge Representation, Module 3

Simone Persiani - 00984854

September 2021

1 Introduction

The present work consists in a deliverable for the exam of *Fundamentals of AI and Knowledge Representation (module 3)*. The aim of my project is to deploy a Bayesian Network model for the analysis of a dataset containing information about F1 races ranging from year 2007 to year 2020 (both included).

I exploited a Python library called **pgmpy** as a tool to build a Bayesian Network which models statistical dependencies between variables related to the realm of Formula One races. The network, including both its structure (DAG) and its internal parameters (CPTs), is entirely learned from data without the need of any prior knowledge about this particular domain.

2 The dataset

I won't explain step by step the full process I followed for data preparation since it would be out of scope for this course. I'll limit myself to list some important remarks about the data which has been collected.

2.1 A big limitation

The reader must keep in mind that the aim of this project is only to model the F1 race history starting from 2007 up to the last completed season (2020, at the time of writing). This means that only the *modern era* of this sport will be considered, disregarding all of the previous F1 history (with the first official F1 Championship race held in 1950).

This is surely a limitation the reader must be aware of. For example, the entire era of absolute dominance of Michael Schumacher at Scuderia Ferrari is kept out.

This choice was made to factor out most of the regulatory changes that have been introduced from the first ever championship onwards, which would have made really difficult to compare data coming from very different epochs. For example, the VSC (Virtual Safety Car) was officially introduced only starting from the 2015 season. Moreover, the qualifying format drastically changed during the 2006 season when the currently adopted *knockout* system was introduced for the first time.

2.2 Sources

Most of the data was taken from a Kaggle dataset maintained by Rohan Rao, an Indian sudoku and puzzle solver. Data contained in it was originally collected from the Ergast REST API. It consists in a set of CSV files containing a lot of attributes (most of which were discarded in order to simplify the final dataset).

Data concerning weather, red flags and (Virtual) Safety Car stints was manually collected by myself from *wikipedia.org* and *www.statsf1.com*. I had to create from scratch a CSV file containing the related attributes and data: it was later merged with the Kaggle dataset described above.

Being more specific, information was drawn from the following pages:

- List of red-flagged Formula One races (en.wikipedia.org)
- Italian Wikipedia articles dedicated to F1 Championships of each considered year, for example:
  - 2007 F1 Championship (it.wikipedia.org)
  - 2012 F1 Championship (it.wikipedia.org)
  - 2017 F1 Championship (it.wikipedia.org)

- *lap-by-lap* tables from STATS F1 for each race, for example:
  - Italy Grand Prix (2009)
  - Monaco Grand Prix (2013)
  - Britain Grand Prix (2015)

3 Learning the model

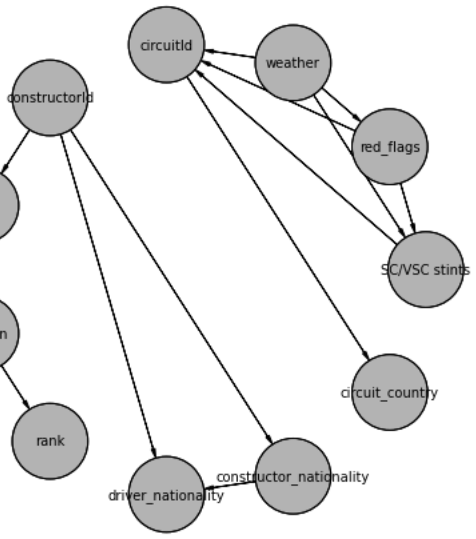
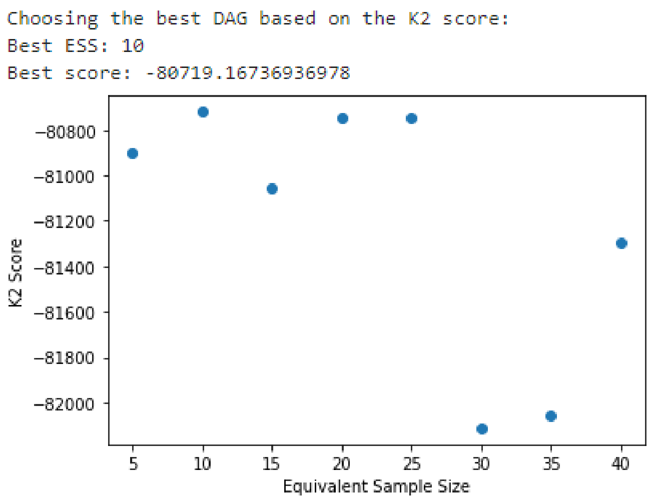


Figure 1: The DAG structure automatically learnt from data.

The Bayesian Network modelling the dataset was entirely learned from data. Firstly, the DAG structure was learned by means of a *hill climbing search with tabu list* algorithm using the BDeu (Bayesian Dirichlet equivalent uniform) as its scoring function. In order to avoid a completely arbitrary choice of the *equivalent_sample_size* parameter of the BDeu score, I repeated the learning process using different values and then I chose the one that led to the higher K2 score (which is nonparametric).

```python
for cur_ess in [5, 10, 15, 20, 25, 30, 35]:
    cur_score = BDeuScore(data=data, equivalent_sample_size=cur_ess)
    cur_dag = HillClimbSearch(data=data).estimate(scoring_method=cur_score,
                                                  max_iter=50,
                                                  tabu_length=100,  # default value
                                                  epsilon=1e-4,  # default value
                                                  show_progress=True)
```



```
Choosing the best DAG based on the K2 score:
Best ESS: 10
Best score: -80719.16736936978
```

Figure 2: K2 score comparison between DAG structures learned using different *equivalent_sample_size* values.

I then had to build a Bayesian network model starting from the chosen DAG structure and automatically learning from data its internal parameters (namely, the entries of the Conditional Probability Table for each node).

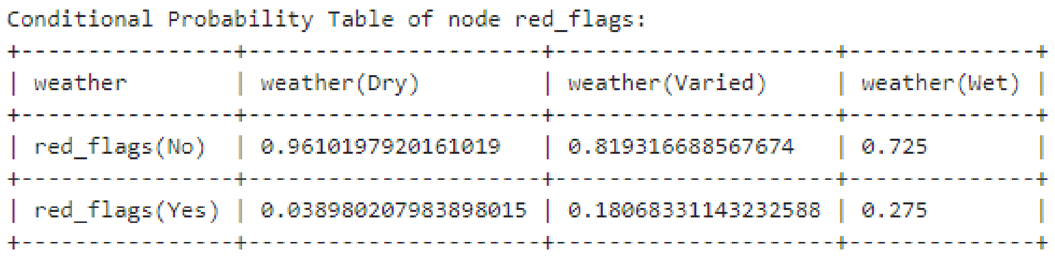
```python
model = BayesianModel(dag.edges())
model.fit(data, estimator=BayesianEstimator, prior_type="BDeu", equivalent_sample_size=ess)
```

Parameters were learned by means of the Bayesian Parameter Estimation method, which was preferred to the simpler Maximum Likelihood Estimation technique because it better avoids overfitting: actually, it regularizes the estimated probabilities by adding *pseudo state counts* to the real counts before normalization. In particular, I chose to use a BDeu prior with an *equivalent_sample_size* equal to the one used to learn the DAG structure.

4 Network analysis

The network was analysed using the methods made available by the **pgmpy** library.

```python
# CPT
print("Conditional Probability Table of node red_flags:")
print(model.get_cpds("red_flags"))
```



```
Conditional Probability Table of node red_flags:
+----------------+----------------------+---------------------+--------------+
| weather        | weather(Dry)         | weather(Varied)     | weather(Wet) |
+----------------+----------------------+---------------------+--------------+
| red_flags(No)  | 0.9610197920161019   | 0.819316688567674   | 0.725        |
+----------------+----------------------+---------------------+--------------+
| red_flags(Yes) | 0.038980207983898015 | 0.18068331143232588 | 0.275        |
+----------------+----------------------+---------------------+--------------+
```

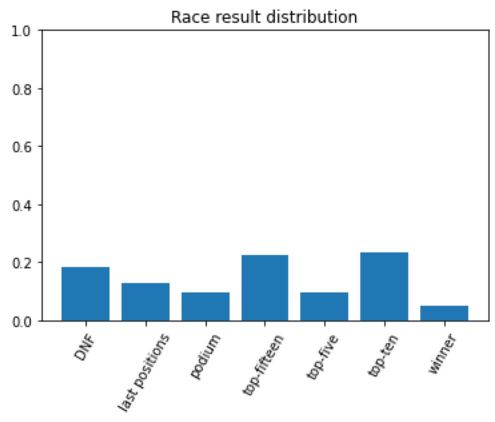
```python
# Children
print("Children of node constructor_nationality:")
print(model.get_children("constructor_nationality"))

# Predecessors
print("Predecessors of node grid:")
print(list(model.predecessors("grid")))

# Successors
print("Successors of node grid:")
print(list(model.successors("grid")))

# Markov Blanket
print("Markov Blanket of node constructorId:")
print(model.get_markov_blanket("constructorId"))

# Local Independencies
print("Local independencies of node weather:")
print(model.local_independencies("weather"))

# Active Trails
print("Active trails between node weather given grid, circuit_country and driver_nationality:")
print(model.active_trail_nodes("weather", observed=["grid", "circuit_country", "driver_nationality"]))
```

5 Queries

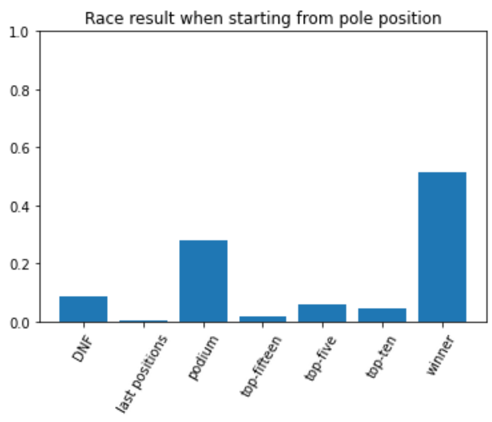
Lastly, I tried to execute some queries on the model. Since the obtained Bayesian Network didn't look too complex, as it can be easily seen in Figure 1, I could afford to compute exact inferences by using the Variable Elimination algorithm.

I started by investigating the relationship between the *position* and *grid* nodes, where the former represents the position of a driver at the end of a race while the latter represents the position from which a driver started a race. The dependency between these two variables turned out to even stronger than I expected, showing how critical is the qualifying result for the race outcome. Some results are plotted below in Figure 3.

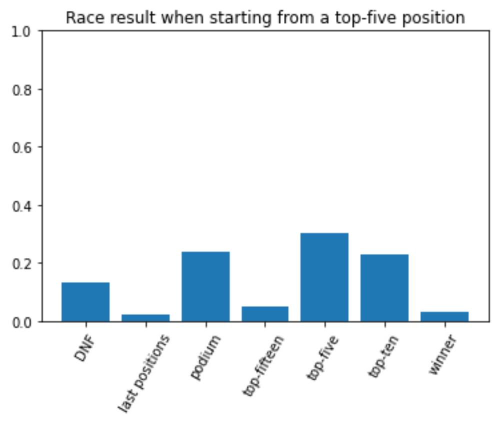
In order to choose their race strategy, F1 teams must take into consideration the likelihood of a red flag or (more likely) a Safety Car deployment. As an example, they could estimate such likelihood by considering the



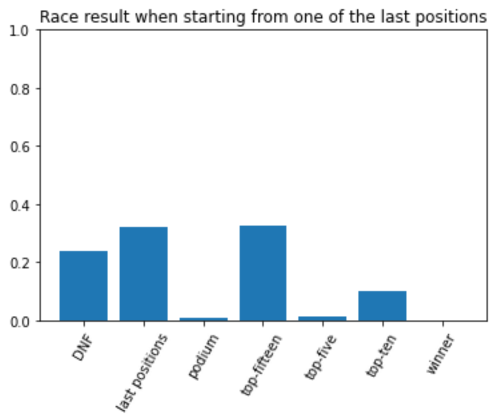
(a)



(b)



(c)



(d)

Figure 3: Query results: probability of position given grid

weather and the circuit at which the race is held (some circuits had historically more frequent SC deployments, either because of changes in the weather or because of big car crashes). The Maximum A Posteriori query is the perfect tool in this case:

```python
print(inference.map_query(variables=['red_flags', 'SC/VSC stints'],
                          evidence={'circuit_country': 'Belgium', 'weather': 'Varied'},
                          show_progress=False))
```

```
{'red_flags': 'No', 'SC/VSC stints': 2}
```

This result means that the most probable thing to happen during a race held in Belgium in varied weather is that no red flag is waved but that the SC is even deployed two times!

I then used some other MAP query to find out some curious stats:

```python
# What constructor is more likely to have at least a car not being able to finish the race?
print(inference.map_query(variables=['constructorId'], evidence={'position' : 'DNF'}))

# What constructor is more likely to obtain the fastest lap of the race?
print(inference.map_query(variables=['constructorId'], evidence={'rank' : 'fastest lap'}))

# What circuit is more likely not to be hit by rain for the entire duration of the race?
print(inference.map_query(variables=['circuitId'], evidence={'weather' : 'Dry'}))
```

The answers are, in the right order:

- *Toro Rosso (now known as AlphaTauri)*;
- *Mercedes*;
- the *Circuit de Barcelona-Catalunya* in Spain.