Hochschule Bremen
City University of Applied Sciences

HSB

Building, visualizing and classifying a NoSQL time series data store using InfluxDB2, Python and Grafana

Big Data & Machine Learning

Philipp Moritzer, 21.07.2022

Table of contents

1. Introduction

This project‘s goals

Introduction

- Big data, e.g. IoT data or real-time analytic data are new types of data
  - ➢ As a result, new methods of storing this data have emerged

- Almost all streaming data, particulary IoT data and application monitoring, have a timestamp
  - ➢ thus is time series data

- This project goes on to demonstrate how time series databases can be used in terms of:
  - Differences to to traditional relational databases
  - Why time series databases are used
  - How to build a project on top of them

- The project:
  - Storing data in InfluxDB
  - Visualizing and classifying the data
  - Python, Grafana, InfluxDB2, Naive Bayes

2. Fundamentals

Understanding NoSQL databases, time series databases, InfluxDB and its capabilites

NoSQL basics

- An approach to database design that excludes traditional relational database management systems
- Efficient for specific data instead of abstracting the underlying data structure
- No need to use a predefined schema
  - ➢ Resulting in simpler design, horizontal scaling and greater control over data
- Distributed data storage
  - CAP-Theorem: Only two out of three attributes: consistency, availability, partition tolerance
  - Depending on the applications‘ needs

[1]

NoSQL vs Relational Databases

- Share the same basic goal: Store and retrieve data, coordinate changes.

SQL	**NoSQL**
Abstraction through relational principle	No data abstraction (data-specific)
SQL Query Language	Custom query language
Lots of application code needed when distributing	Distributed by design
Pre-defined Schema	No Schema
Simple coordination properties	Might get complex fast
Inefficient when handling lots of unstructured data	Scalable by design, can handle loads of data

Time Series Databases

- Grouping of values organized by time
- Any event recorded over time (regular or irregular) is considered time series data
- Designed to deal with high volume measurement data
- Solve 3 major characteristics of data:
  - Exceptionally high volume
  - Natural time order
  - The entire set of data being more valuable than individual records
- Optimizes on frequent writes, merging data and constructing sums and averages
- Retention period
- Optimzed query language
- Solving the problem with SQL: Not designed to do frequent deletes

Popular time series databases: influxdb [2] Timescale [4] Prometheus [3]

InfluxDB basics

- Most popular time series database
- Created 2013, Version 2 in 2020 with new Query Language and skyrocketing popularity
- Standalone database system
- Advertised for solving following problems:
  - Collecting and storing IoT and real-time data
  - Analyzing data
  - High performance (million datapoints per second)
  - Timestamps in nanoseconds range

InfluxDB basics - Flux query language

- Includes a query language for querying, analyzing and acting on data
- Can perform various operations but a general query looks as follows:

```
from(bucket)
        |> range (start: x, stop: y)
        |> filter
```

- The pipe symbol |> marks pipe forward data and every function or expression that follows takes the former expression as an input
- A bucket is used for storing time series data in InfluxDB which will be choosen as source first
- A time range to select is required
- One can filter the data based on their requirements
- Offers functions for reducing, summing, interpreting or mapping data
  - ➢ Therefore Flux can be used for data analytics

InfluxDB basics - Flux query language (example)

```
from(bucket: “bird-migration“)
        |> range(start: 2021-01-01T00:00:00Z, stop: 2021-01-01T12:00:00Z)
        |> filter (fn: (r) => r._measurement == "location" and r._field == "lat"
and r.lat == "lat")
```

Comparing InfluxDB to other time series databases

Timescale [4]

- Based on PostgreSQL
- Uses SQL as query language
- Relational data model
- Inferior performance to InfluxDB

Prometheus [3]

- Query Language PromQL
- More features for monitoring purposes
- Less support for analytics or machine learning
- Only milliseconds stamps vs InfluxDB‘s nanoseconds
- Less resource usage

Cloud: AWS (Amazon timestream) or Azure(Azure Time Insights) prefer to use their cloud native database due to the infrastructure

Classification using Naive Bayes

- Probabilistic classifier that can predict a probable outcome of a class if a field is given.
- Based on the Bayes Theorem:

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

- A distribution over a set of classes is calculated given an observation of an input
- The classifier can the be trained to determine which class has the highest probability

$$P(Class : | : Field)$$

- The class can be predicted given a field input
- Example:

$$P(Winter|Tropes) = 0.7$$

3. Project

Introducing the project implemented

Project - Overview

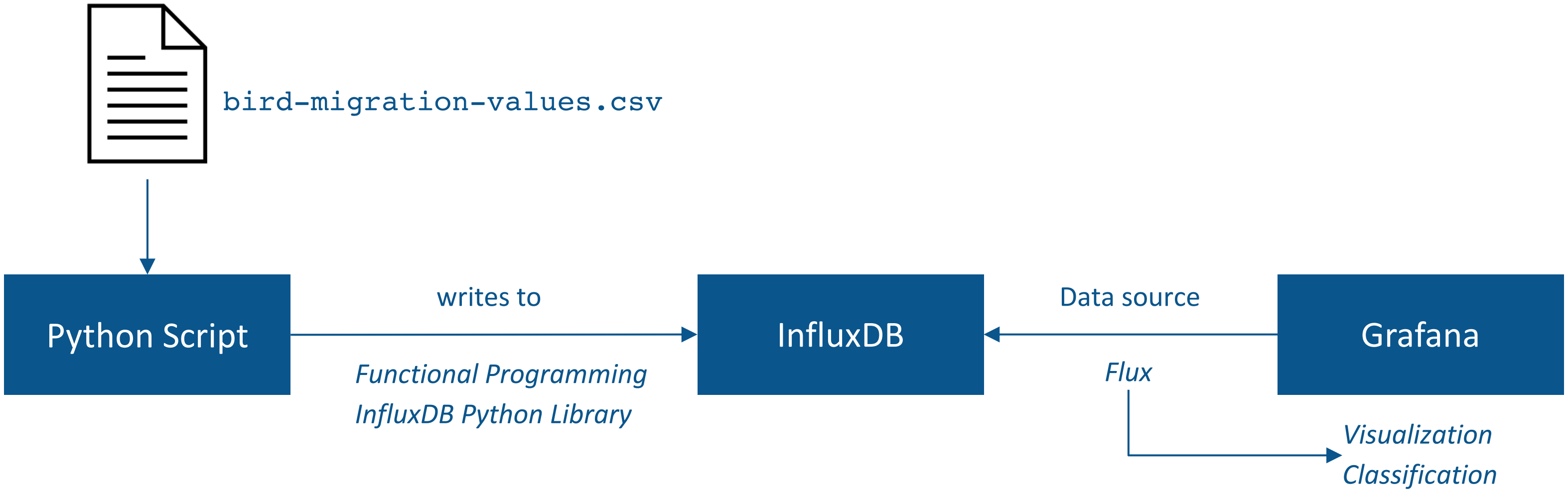
- Process a large amount of time series data
- NoSQL with InfluxDB
- A Grafana Dashboard is constructed to visualize the data
- The data (CSV) will be parsed using Python and processed to InfluxDB
  - Using Functional Programming
  - Using the Python client library provided by InfluxDB
- Classification with Flux

Project – The dataset

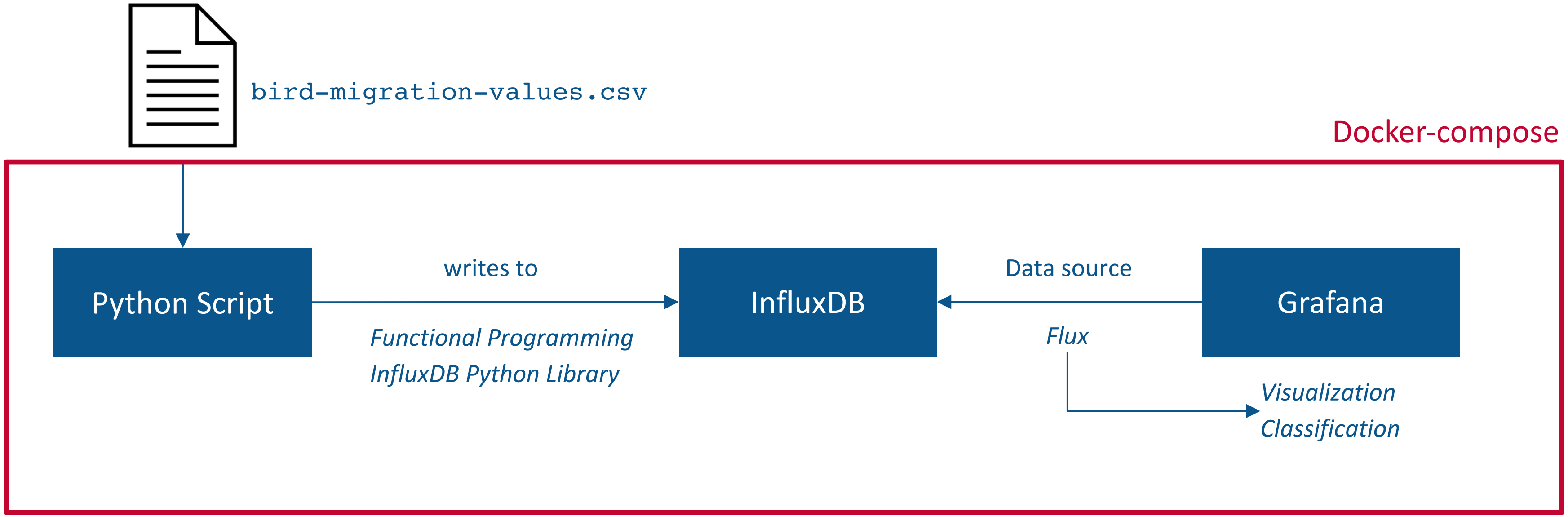
- Values that represent measured bird locations over a series of time
- Stored as a CSV-file
- Contains 89869 measurements
- Time range from 2009-2016
- 49 individual birds

event-id	visible	timestamp	location-long	location-lat	manually-marked-outlier	visible	sensor-type	individual-taxon-canonical-name	tag-local-identifier	individual-local-identifier	s
1082620685	TRUE	2009-05-27 14:00:00.000	24.58617	61.24783		TRUE	gps	Larus fuscus	91732	91732A	N
1082620686	TRUE	2009-05-27 20:00:00.000	24.58217	61.23267		TRUE	gps	Larus fuscus	91732	91732A	N
1082620687	TRUE	2009-05-28 05:00:00.000	24.53133	61.18833		TRUE	gps	Larus fuscus	91732	91732A	N
1082620688	TRUE	2009-05-28 08:00:00.000	24.582	61.23283		TRUE	gps	Larus fuscus	91732	91732A	N
1082620689	TRUE	2009-05-28 14:00:00.000	24.5825	61.23267		TRUE	gps	Larus fuscus	91732	91732A	N
1082620690	TRUE	2009-05-28 20:00:00.000	24.58617	61.24767		TRUE	gps	Larus fuscus	91732	91732A	N
1082620691	TRUE	2009-05-29 05:00:00.000	24.586	61.24767		TRUE	gps	Larus fuscus	91732	91732A	N
1082620692	TRUE	2009-05-29 08:00:00.000	24.58617	61.24767		TRUE	gps	Larus fuscus	91732	91732A	N
1082620693	TRUE	2009-05-29 14:00:00.000	24.5865	61.2475		TRUE	gps	Larus fuscus	91732	91732A	N
1082620694	TRUE	2009-05-29 20:00:00.000	24.56967	61.23883		TRUE	gps	Larus fuscus	91732	91732A	N
1082620695	TRUE	2009-05-30 05:00:00.000	24.58667	61.2475		TRUE	gps	Larus fuscus	91732	91732A	N
1082620696	TRUE	2009-05-30 08:00:00.000	24.58617	61.2475		TRUE	gps	Larus fuscus	91732	91732A	N
1082620697	TRUE	2009-05-31 14:00:00.000	24.587	61.2475		TRUE	gps	Larus fuscus	91732	91732A	N
1082620698	TRUE	2009-05-31 20:00:00.000	24.47967	61.2105		TRUE	gps	Larus fuscus	91732	91732A	N
1082620699	TRUE	2009-06-01 05:00:00.000	24.5825	61.233		TRUE	gps	Larus fuscus	91732	91732A	N
1082620700	TRUE	2009-06-02 05:00:00.000	24.55183	61.22933		TRUE	gps	Larus fuscus	91732	91732A	N
1082620701	TRUE	2009-06-02 14:00:00.000	24.49183	61.217		TRUE	gps	Larus fuscus	91732	91732A	N
1082620702	TRUE	2009-06-02 20:00:00.000	24.442	61.19		TRUE	gps	Larus fuscus	91732	91732A	N

Project - Overview

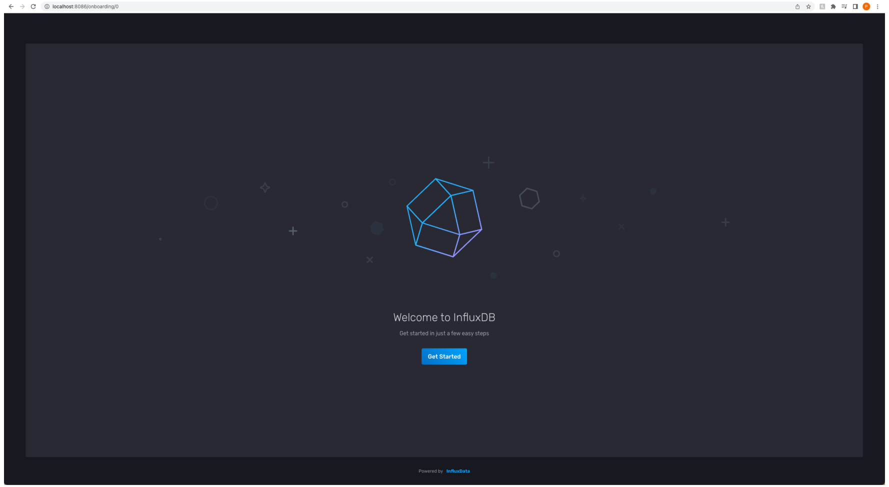


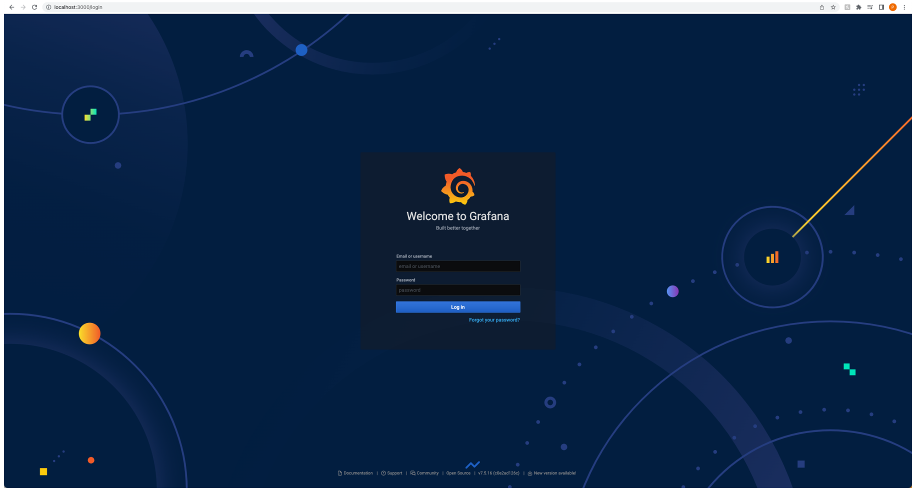
Project - Overview



Project – Setting up the infrastructure

- Local instances of InfluxDB and Grafana using Docker/docker-compose
- Using official Docker images for InfluxDB and Grafana
  - Therefore no installation necessary





Project – Importing the data

- Python application
- InfluxDB Client Library
  - Installed with pip
- Functional Programming
- Batch processing
- Database connection retry mechanism
- Dockerized

Project – Importing the data

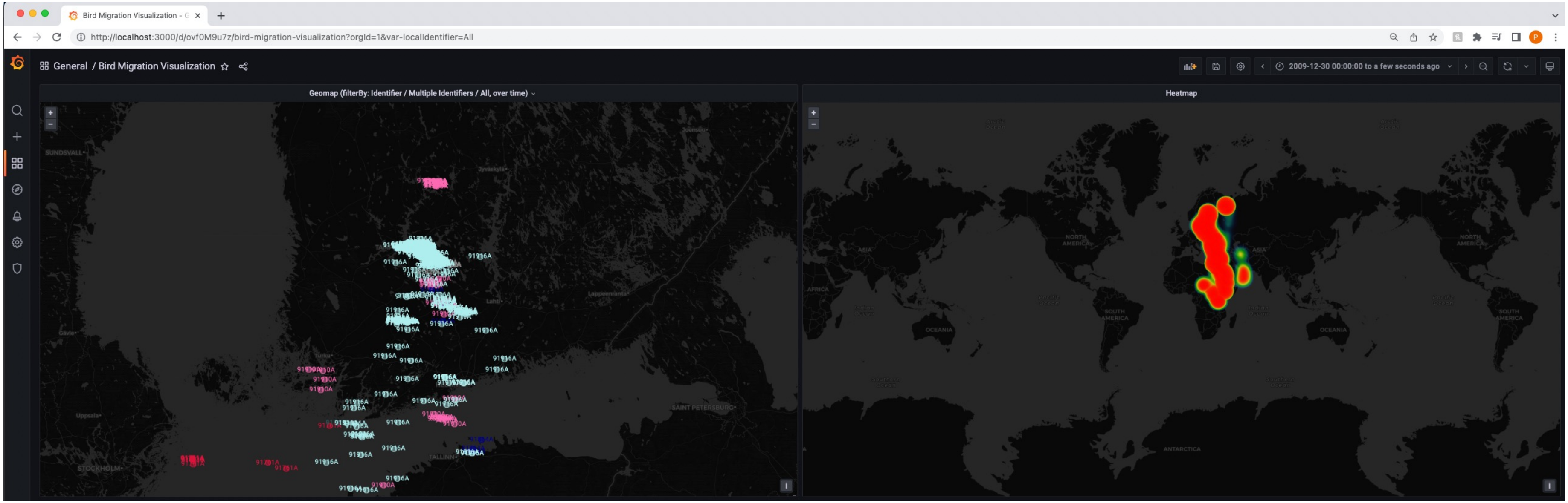
```python
def parse_row(row: OrderedDict):

    return Point("migration-point").tag("type", "migration-value").measurement("migration") \
        .field("event-id", row['event-id']) \
        .field("lon", Decimal(row['location-long'])) \
        .field("lat", Decimal(row['location-lat'])) \
        .field("manually-marked-outlier", row['manually-marked-outlier']) \
        .field("individual-taxon-canonical-name", row['individual-taxon-canonical-name']) \
        .field("tag-local-identifier", row['tag-local-identifier']) \
        .field("individual-local-identifier", row['individual-local-identifier']) \
        .time(row['timestamp'])

data = rx \
    .from_iterable(DictReader(open('migration_original.csv', 'r'))) \
    .pipe(operators.map(lambda row: parse_row(row)))
```

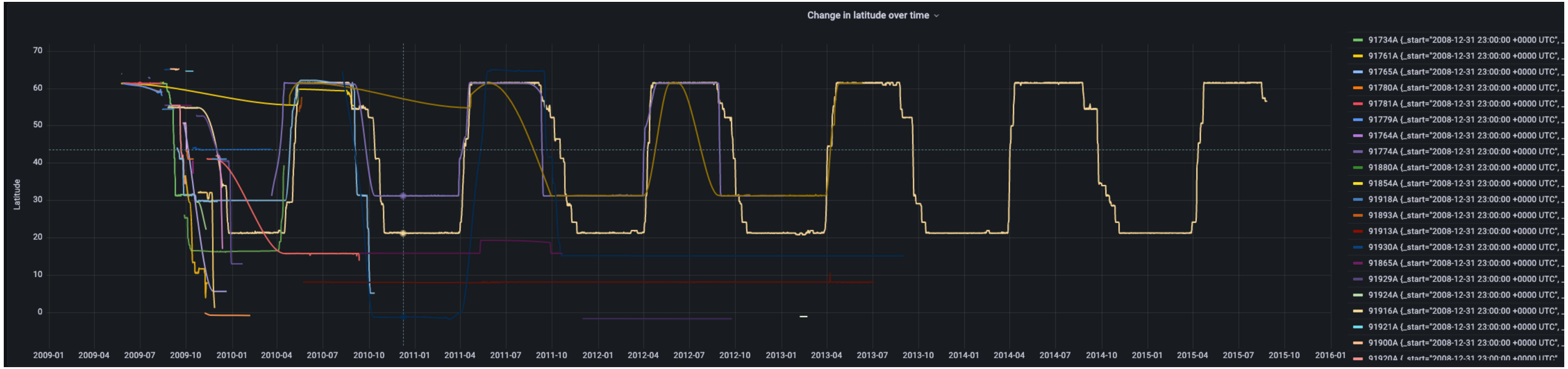
Project – visualization – GeoMap & Heatmap

- Grafana GeoMap & Heatmap – visualizing the data entries on a map
- Filterable by time and by bird identifier



Project – visualization – Latitude over time

- Grafana Time Series Chart
- Filterable by time and by bird identifier



Project – classification

- Predicting the season based on the bird's location
- Mapping the data
- Dataset only contains latitude and the timestamp as possible identifier which have a really broad range of values
  - ➢ Values are mapped: Months to season and latitude to climate zone

Months	Season
12-02	Winter
03-05	Spring
06-08	Summer
09-11	Autumn

Latitude	Location
60-90	Cold
40-60	Mild
23.5-40	Subtropical
0-23.5	Tropical

Project – classification – example calculation

- Transforming the data to binary with class/field pairs as training data

Training data

_season	cold	mild	subtropical	tropical
spring	1	0	0	0
spring	1	0	0	0
spring	1	0	0	0
spring	1	0	0	0
spring	1	0	0	0
spring	1	0	0	0
spring	1	0	0	0
spring	1	0	0	0
spring	1	0	0	0
spring	1	0	0	0
spring	1	0	0	0
spring	1	0	0	0
spring	1	0	0	0
spring	1	0	0	0

Project – classification – example calculation

- Transforming the data to binary with class/field pairs as training data

- $P(winter \mid tropes) = \frac{P(winter) * P(tropes|winter)}{P(tropes)}$
- $P(winter) = \frac{winter\_season\_entries}{all\_entries}$
- $P(tropes) = \frac{tropes\_entries}{all\_entries}$
- $P(tropes\ |winter) = \frac{tropes\_in\_winter}{winter\_season\_entries}$

- Sample data:

$$
\begin{array}{c}
all\_entries = 50 \\
Winter\_season\_entries = 21 \\
tropes\_entries = 20 \\
tropes\_in\_winter = 18
\end{array}
$$

- $P(winter \mid tropes) = \frac{0.42 * 0.9}{0.4}$ **= 0.945 = 94.5%**
- $P(winter) = \frac{21}{50} = 0.42$
- $P(tropes) = \frac{20}{50} = 0.4$
- $P(tropes\ |winter) = \frac{18}{20} = 0.4$

Project – classification – example calculation

- Preprocess the training data to get every value needed for the calculation

Mapped occurences for calculation

_class	_field	all_entries	autumn_season_entries	cold_entries	field_in_class_count	mild_entries	spring_season_entries	subtropical_entries	summer_season_entries	tropical_entries	winter_season_entries
winter	tropical	13974	3240	3283	2078	1252	3754	3175	3574	6211	3406
winter	subtropical	13974	3240	3283	1307	1252	3754	3175	3574	6211	3406
winter	mild	13974	3240	3283	0	1252	3754	3175	3574	6211	3406
winter	cold	13974	3240	3283	0	1252	3754	3175	3574	6211	3406
spring	tropical	13974	3240	3283	1675	1252	3754	3175	3574	6211	3406
spring	subtropical	13974	3240	3283	723	1252	3754	3175	3574	6211	3406
spring	mild	13974	3240	3283	377	1252	3754	3175	3574	6211	3406
spring	cold	13974	3240	3283	968	1252	3754	3175	3574	6211	3406
summer	tropical	13974	3240	3283	1160	1252	3754	3175	3574	6211	3406
summer	subtropical	13974	3240	3283	3	1252	3754	3175	3574	6211	3406
summer	mild	13974	3240	3283	196	1252	3754	3175	3574	6211	3406
summer	cold	13974	3240	3283	2215	1252	3754	3175	3574	6211	3406
autumn	tropical	13974	3240	3283	1298	1252	3754	3175	3574	6211	3406
autumn	subtropical	13974	3240	3283	1142	1252	3754	3175	3574	6211	3406

Project – classification – example calculation

- Calculate the trained classifier for each Class given field:

Naive Bayes Classifier

_class	_field	p_class	p_field	p_field_class	probability
winter	tropical	0.225	0.393	0.594	0.341
winter	subtropical	0.225	0.229	0.385	0.379
winter	mild	0.225	0.127	0.0155	0.0276
winter	cold	0.225	0.248	0	0
spring	tropical	0.229	0.393	0.440	0.257
spring	subtropical	0.229	0.229	0.190	0.190
spring	mild	0.229	0.127	0.0990	0.179
spring	cold	0.229	0.248	0.269	0.249
summer	tropical	0.268	0.393	0.260	0.178
summer	subtropical	0.268	0.229	0.000673	0.000789
summer	mild	0.268	0.127	0.0877	0.186
summer	cold	0.268	0.248	0.651	0.705
autumn	tropical	0.277	0.393	0.320	0.225
autumn	subtropical	0.277	0.229	0.356	0.430
autumn	mild	0.277	0.127	0.278	0.607
autumn	cold	0.277	0.248	0.0411	0.0459

4. Live Demonstration

Showing the Project and its results

5. Conclusion

Reflecting on the results

Conclusion

- InfluxDB is an excellent tool for dealing with large datasets of time series data
- Suitable for real-time analytics, also when combining it with its Python client library
- The query language Flux is capable of performing extensive data analytics
- Integration with Grafana and the visualization works very well

Outlook

- Real-time streaming use case
- Real-time analytics
- Distribute the data store
- Streaming IoT data

Repository

- View the project at:

https://github.com/philippmoritzer/bd-ml-project

Hochschule Bremen
City University of Applied Sciences

HSB

Thank you!

Sources 1

- [1] Brad Dayley. Sams Teach Yourself NoSQL with MongoDB in 24 Hours, Video Enhanced Edition. O'REILLY. 2014.
- [2] Kasun Idrasiri, Sriskandarajah Suhothayan. Design Patterns for Cloud Native Applications. O'REILLY. 2021.
- [3] CloudLab. NoSQL - CAP Theorem. Author unknown. Date unknown. URL: https://cloudxlab.com/assessment/displayslide/345/nosql-cap-theorem#:~:text=NoSQL%20can%20not%20provide%20consistency,Consistency%2C%20Availability%20and%20Partition%20Tolerance. (visited: 10.07.2022, 20:15)
- [4] Ted Dunning, Ellen Friedman. Time Series Databases: New Ways to Store and Access Data. O'REILLY. 2014.
- [5] Joe Reis, Matt Housley. Fundamentals of Data Engineering: Plan and Build Robust Data Systems. O'REILLYs. 2022.
- [6] Paul Dix. Why Build a Time Series Data Platform?. db-engines. 2017. https://db-engines.com/en/blog_post/71 (visited: 10.07.2022, 20:15)
- [7] Kovid Rathee. The case for using timeseries databases. 2021. URL: https://towardsdatascience.com/the-case-for-using-timeseries-databases-c060a8afe727 (visited: 10.07.2022, 20:15)
- [8] db-engines. InfluxDB System Properties. 2022. URL: https://db-engines.com/en/system/InfluxDB (visited: 10.07.2022, 20:15)
- [9] influxdata. influxdata - Documentation. 2022. URL: https://docs.influxdata.com/ (visited: 10.07.2022, 20:15)
- [10] influxdata. Get started with Flux. 2022. URL: https://docs.influxdata.com/influxdb/cloud/query-data/get-started/ (visited: 10.07.2022, 20:15)

Sources 2

- [11] Rohan Sreerama. A Deep Dive into Machine Learning in Flux: Naive Bayes Classification. 2020. URL: https://www.influxdata.com/blog/deep-dive-into-machine-learning-in-flux-naive-bayes-classification/ (visited: 10.07.2022, 20:15)
- [12] Igor Bobriakov. Prometheus vs InfluxDB. 2020. URL: https://www.metricfire.com/blog/prometheus-vs-influxdb/ (visited: 10.07.2022, 20:15)
- [13] db-engines. System Properties Comparison InfluxDB vs. Prometheus vs. TimescaleDB. 2022, URL: https://db-engines.com/en/system/InfluxDB%3BPrometheus%3BTimescaleDB (visited: 10.07.2022, 20:15)
- [14] United Manufacturing Hub. Why we chose timescaleDB over InfluxDB. 2022, URL: https://docs.umh.app/docs/concepts/timescaledb-vs-influxdb/
- [15] Team Magic. Building a Naive Bayes classifier using Flux. 2020. URL: https://github.com/RohanSreerama5/Naive-Bayes-Classifier-Flux/blob/master/Naive%20Bayes.pdf (visited: 10.07.2022, 20:15)
- [16] Rohan Sreerama. Naive-Bayes-Classifier-Flux. 2020. URL: https://github.com/RohanSreerama5/Naive-Bayes-Classifier-Flux (visited: 10.07.2022)

Image Sources (all accessed on: 17.07.2022, 15:00)

- [1] Top 10 noSQL Databases: https://ares.decipherzone.com/blog-manager/uploads/ckeditor_Top%2010%20NoSQL%20Databases%20in%202022.png
- [2] InfluxDB: https://dbdb.io/db/influxdb/revisions/7
- [3] Prometheus: https://blogs.sap.com/wp-content/uploads/2018/01/prometheus.png
- [4] TimescaleDB: https://3rdman.de/wp-content/uploads/Timescale.png