Eötvös Loránd University

Faculty of Informatics

Department of Information Systems

Framework for human mobility prediction based on CDR data

Sándor Laki, PhD

Assistant professor

Ákos Rudas

PhD student

Gábor Brájer

Computer Science MSC

Budapest, 2017

Table of contents

Abstract

In this master thesis anonymized empirical telecommunication data will be explored and analyzed with both traditional statistical methods and machine learning algorithms in order to get a meaningful insight on human mobility patterns in the targeted area and how predictable this movement is. With better understanding of this data we might have a clearer view on the evolving society and ever-changing behaviour of mass mobility. The goal is twofold: make as accurate predictions as possible based on historical telco data and find interesting discoveries in the form of outliers, anomalies and unforeseen mobility trends.

Enormous amount of telco data is being generated every day, so performance and scalability were in focus while designing the solution. For this reason state-of-the-art big data environment (Apache Spark [5]) was used to perform the data analysis. The telco data consists of entries from all around Hungary. Telco data from all around the country, the capital (with suburbs and downtown area) and an important choke point (Ferihegy Airport) will be examined separately.

The analysis consists of how far people travel every day [1], show longest paths, visualize how masses move around up to hourly precision and finally evaluate different prediction models of mass mobility. The pipeline of tasks are designed to continuously feed information for the next steps and ensure seamless distribution of data and computation in an imagined clustered environment, since a single bottleneck may slow down the entire process. This thesis intends to show the motivation, process, technologies used, and results gained.

KEYWORDS telecommunication data, crowd mobility prediction, big data analysis, large-scale and distributed computing

“The world is one big data problem.” – Andrew McAfee

1. Introduction

Telecommunication is undoubtedly one of the biggest businesses today. All over the world - with a few exceptions - mobile phones became the number one companion for people and it does no longer hold for just modern societies. Everyone is instantly accessible via a call or text message. No matter if you are home, in the office or travelling - there is close to no limitations when or where you can use your mobile phone. Supporting this statement let's see a visualization of cellular coverage of the biggest telco provider in Hungary [3]. Including all wireless mobile telecommunication technologies (2G, 3G, 4G, ...) the coverage is basically 100% and this is only one provider.

With the infrastructure and number of phone users we can expect a lot of cellular activity. Mobile providers are tracking and persist mobile events for a limited amount of time for a couple of reasons:

(1) paid services and billing are calculated based on usage
(2) track service availability and identification of bottlenecks
(3) find anomalies or misconfiguration

Recently we can add a fourth point to our list and that is *big data analytics*. But why would mobile data providers be interested in analyzing their data and share valuable business information with researchers having the risk that it might leak out?

1.1 Motivation

There are plenty of useful findings that might come out of big data analytic and has business value. Predictive analysis might help to predict outages, load spikes, smart adjustments of tower capacity, future tower load and gives a helping hand to achieve better overall service quality and increased uptime. Anomaly detection algorithms [2] can detect and pinpoint misconfigurations in related IT or telco network systems. Characterization of human mobility patterns helps with crowd avoidance, optimal route planning and smart cities for traffic management.

What do mobile providers have to do to have access to such knowledge and how much does it cost them? This is the best news for providers and researchers: mobile CDR (*Call Detail Record*) data is already being continuously collected by mobile operators. Working with CDR is cheap and frequent way to collect data for research purposes and it is available in large scales. There is no need to add dedicated measure systems, fill in surveys or force users to install additional applications.

Complementing the already mentioned reasons providers are obliged by law to store call logs for a specified amount of time. These data may come in handy for forensic analysis, fraud detection and be used to assist ongoing investigations or to reconstruct criminal events and unveil the underlying criminal network [4].

In addition to the CDR data that will be used for the analysis involvement of other publicly available external data sources could be used to improve predictive power and construct more accurate models. Originally the plan was to include weather data metrics like temperature, air pressure or moisture. Actual weather could potentially have a huge impact on our everyday travelling trajectory and distance. Unfortunately, due to the lack of time it did not make it to the final analytic models.

1.2 Objectives

The objective of this thesis is to discover, analyze and visualize different aspects of human mobility patterns and - with the results gained - try to predict these activities in arbitrary time frames. There is huge variety in analyzing such data and it is a highly creative task.
The main topics that will be explored here are the following:

(1) How far people travel each day
(2) Show longest paths taken per day
(3) In case of an event (football match) or a focused area (downtown) how well can we predict crowd at a given location?
(4) Predict crowd density and mobility at a given place and time

The above-mentioned metrics may have a lot of different outcomes based on how we handle the data: we can include weather data, separate weekdays and weekends or rush hours (let’s say Monday-Friday 9 am-5 pm) from non-rush hours, etc. With this in mind there is still a lot of future potential and expandability in it.

Giving a look at the technology side of things the data clearing, transformation, and analysis on raw data is done with Apache Spark (and its extension libraries Spark ML and Spark SQL) and the visual representation is done by a web service that generates views based on data produced by the backend analysis systems. The output data can be plugged directly into the visualization application without any further refinement. Together they represent the CDR mobility prediction framework.

1.3 Privacy concerns

What about subscriber identity protection and route tracking? Should users be concerned that they are endlessly being monitored by the all seeing eye? CDR data is a valuable asset to better understand human mobility and ultimately benefits everyone, but does it break any legal or moral barriers to track the activity of users of the cellular network? Cellular network users are unaware of such background data logging, but still have the right to have their privacy protected. This concern is even more important when data is redistributed to other third parties for further analysis - like this thesis.

Several security measures are taken by mobile operators before they release such data. Firstly all identification attributes (IMSI, IMEI, Phone number, etc.) are replaced with one auto-generated artificial identifier. Secondly this hash is regenerated every day for every user. This means activities of an individual user could be tracked up to one day. Lastly very few attributes are actually needed for the analysis from the CDRs:

- anonymized mobile subscriber identity
- timestamp of the cellular event
- positional data (latitude and longitude)

Another indirect aspect of privacy protection is the fact that CDR data is sparse: log entries are generated in the initialization moment of cellular events (outgoing and inbound call or SMS). Otherwise the location of the user is completely invisible - there is no handover information or continuous data collection. The geographical data points to the cellular tower's location that handled the event and not that of the user, therefore it is only an approximation of the real position.

With the motivation being clear, objectives set and privacy concerns assured let's jump right into the analysis of the initial sample set that was made available for exploration.

2. CDR data

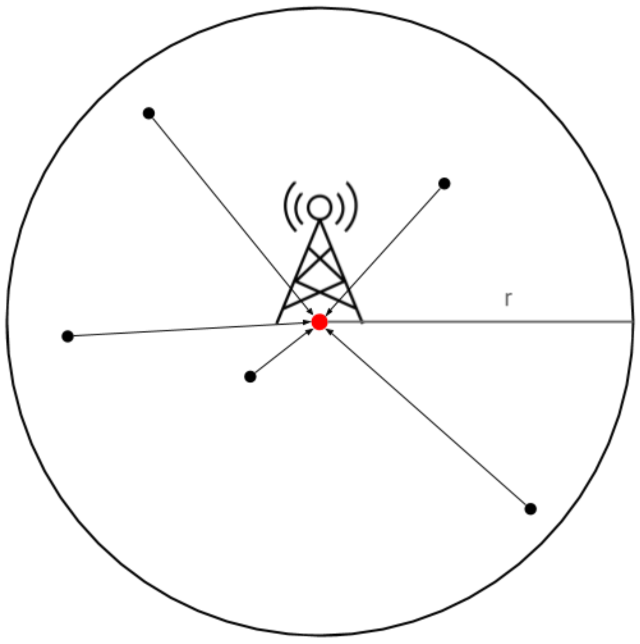
CDR (*Call Detail Record*) data is continuously being collected by major mobile service operators from all around the world for infrastructure, billing and security concerns and are produced by telecommunication transactions. It contains metadata information about the event such as starting time, duration (0 in case of text messages), type (call, SMS, MMS or mobile data), identifier and spatial location of both parties involved in the exchange and does not include exchange contents. The structure is roughly similar everywhere therefore the same analyzers could be used with data from different providers or gathered data from more providers with minimal effort.

The two most defining attribute from the CDR dataset will be time and location data. The time data format is ISO-8601 compliant in CET timezone. Spatial data is a little bit trickier. First of all location is **cell based** and not phone based. This means spatial data will be selected from a fixed set of (cell/tower) locations. A cellular tower may hold multiple number of antennas that point to different compass directions.

2.1 CDR limitations

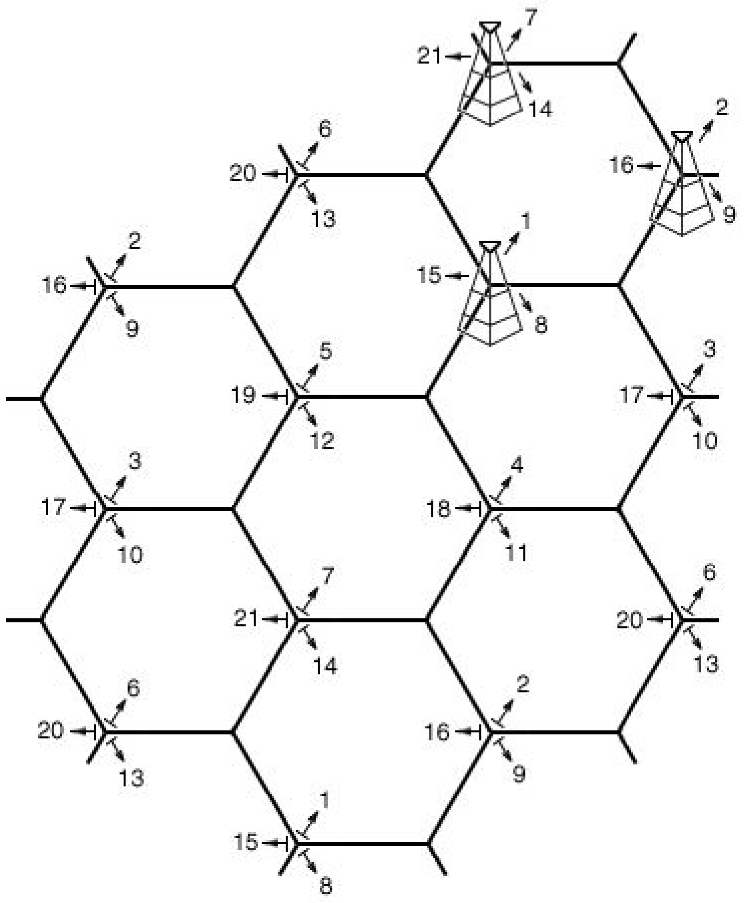
Naturally there are more towers in densely populated areas (cities) and less in rural areas. The towers are also rapidly adjusting their signaling range based on load. If a tower is heavily loaded then it decreases its radius and let another tower take over more clients. The consequence of this ever-changing behaviour is that there is no guarantee that making a telco exchange in the same spot in different times will be handled by the same tower. External noise could also affect this process like weather or receiver electronics properties and interference from other nearby electronic devices. Considering all these factors it can be seen that CDRs may fall prey to a lot of inaccuracy and inconsistency. They are also only collected during the *initialization* of the telco exchange.

Can we safely say that CDRs will be *accurate enough* to give meaningful insight? The data is highly sparse and in rural areas they cover a much larger area with lower resolution, than those located in the cities.



CDR data inaccuracy: black dots are individuals, red dot is the tower's position and r is the current signaling radius of the tower.

In [1] the authors recruited volunteers to log accurate positional data on their phones and compared it to their CDRs from the same time frames. The results showed that the median difference between CDR and actual position is marginal.



Cellular network with directional antennas (source: Wikipedia).

2.2 Poisson distribution

Phone calls arrive with *Poisson* distribution (λ), which means they are completely random with arrival rate λ. Each telco exchange terminates independently and starts with rate λ independently. The traffic intensity is derived from the *arrival rate* and the *mean call duration*. It plays a key role of in modelling such a problem and is an ordinary way to model events initiated by humans. For this reason we need data in large scales, modelling data with small sample size might lead to incorrect conclusions - something that have been experienced during this thesis. Two datasets have been looked at: a heavily sampled dataset (03.10.2016 - 9.10.2016.) of one week and a real dataset of 2 weeks (15.09.2016 - 28.09.2016) with all CDRs recorded. Both of them will be discussed in a sense of basic statistics and how good foundation they are for the predictive models. Sampled set quickly showed its' weaknesses in the predictive models with being very spatial. Firstly let's take a closer look on the sample data and then continue with the real data.

2.3 Data exploration and sample dataset

The telco dataset that was initially available was a minor fraction of the actual dataset - a heavily sampled data of the original set to toy with basically. There were 3 groups of dataset to work with: **CRM** data, **MSC** data and **TAC** data ranging from 3rd of October to 9th of October in 2016 - exactly one week of data separated in different files - both in csv and xls format. Naturally - in all datasets - the IDs were anonymized and rehashed every day by the provider for privacy protection. However, the data from the different datasets could be joined with the artificially generated IDs, since they were the same for the same customer per day. For mobility prediction very few of the accessible attributes were actually needed. The datasets were ordered by day and subscriber ID only. In the end only the MSC dataset holds values that could contribute to the predictions. After testing the algorithms locally with the sampled dataset of one week of data I have moved forward and ran them on unfiltered datasets of 2 weeks on a high performance academic computer. Because of the scalable nature of Apache Spark little to no modifications to the codebase was needed.

The CRM dataset contains subscriber-related information (zip code, sex, age, private or business subscriber, etc.) for users who had telco exchange on that particular day. This dataset was left out from our analysis completely for two reasons: our point of interest is human mobility and crowd, not the composition or characteristics of it. We do not care about the age, sex or the subscription bundle they has. The only characteristic that could have been a valuable asset to the model is the zip code of the users. With that information setting up home-work routes could have been made for instance. Nevertheless, practice shows that considerable percentage of the users do not live at the location suggested by their CRM data, like business subscribers. The second reason is to limit personal details to the minimum and it also worth mentioning that getting more sensitive subscriber data from the provider could have been problematic.

The TAC dataset holds device specific data like TAC (Type Allocation Code), manufacturer, model and other phone specific attributes are found. These informations are borderline useless in our analysis here, so they were discarded, but can be an interesting addition later to the model.

The MSC dataset is the primary playmaker here, where telco event initiations are stored. For every telco exchange two rows are stored: one is the sender/caller side and the other one is the receiver. These two could be joined by another auto-generated identification number, but it is meaningless for now, since we are not interested in the call graph. Most important aspects of this dataset are **location** and **timestamp**. Almost every other important feature may be derived from these.

Imagine location data as hexagonal cells with a given area (see picture above). The hexagons are not perfectly shaped, have the same size or equally distributed. They rapidly change signaling strength based on load. MSC dataset contains **cell id**s and a mapping from cell id to actual spatial coordinates (*latitude* and *longitude*) was also made available. In the beginning only data collected by towers operated by the provider were released, but later data from competitor's towers was acquired (only data from subscribers of the original provider).

Schema of the sample MSC dataset (*CDR*).

Dataset	Date information in *MM.DD* format [*String*]
Subscriber	An active SIM card of a subscriber that is anonymized every day with hashing [*String*]
TAC	TAC - ID of the exact type of the phone [*Integer*]
Type	Type of the telco event (1 - inbound call, 3- outgoing call, 7 - received SMS, 9 - sent SMS) [*Integer*]
Timestamp	Timestamp of the telco event in *YYYY-MM-DD**T**HH:MM:SS* format [*Timestamp*]
Unix	Unix form of the timestamp column [*Long*]
Latitude	Latitude of the tower [*Double*]
Longitude	Longitude of the tower [*Double*]

After initial data exploration feature selection is the next step. For performance and memory limitation reasons every column that has no value have to be dropped right away. TAC column was thrown out along with unix due to redundancy with the timestamp column. The telco type was also discarded, because we are interested in actual crowd and not the way this crowd communicates.

Around ~37k rows were found in this dataset with an error rate of ~0.003% over all rows. The sample set is a controlled set therefore the petite error/missing value rate and no outliers were detected. Much higher variance and error rate is expected with the real dataset - something that must be remembered. The dataset contained 4459 unique subscribers over the week and 2047 unique cell towers involved in the telco exchanges. In the sampled dataset all the data were collected only by towers operated by the provider.
Data transformation was unnecessary for this dataset, since no outliers were detected and - unlike with the real dataset - the cell tower positional data (latitude, longitude) was already wired into the data source.

2.4 Large-scale telecommunication data

The real **MSC** datasets - consisting 2 weeks of telco data - is a bit different from its sampled counterpart. The schema is different - it holds more attributes - and has a larger percentage of invalid values. The latitude and longitude values were absent and the cell was represented with an unique hash. Note that this dataset contains records from other provider's tower data. An external script had to be used to map the hash to latitude and longitude values, but it could not be done for every tower not belonging to the original provider's network and were discarded from the dataset. Most of the new columns present here seemed static values (Network Identification or Device Type) not contributing anything to the analyzers, but increasing the datasets' size. Another new column is the call identification number (*Call ID*) that bundles together an event (call setup, handovers, ...). Appending positional cell data (latitude, longitude) increased the size of the individual datasets by an additional ~30%. The dataset column was not part of the original dataset and was mapped from the timestamp column at runtime.

Stepping up from ~3 MB data to a hefty **~41 GB** (~2,5-3,5 GB per day) data caused some weakness to emerge in certain algorithms used. Half of the analyzers ran without any problem, others were extremely slow or even got stuck. Augmentations had to be done in algorithms and caching strategy in the problematic analyzers. The naive or brutal approaches (mainly $O(n^2)$ algorithms) might work for small datasets, but not for big data. Even in a cluster with powerful computers brute force strategies are not guaranteed to work - a phenomenon that was experienced here.

Schema of the real MSC dataset (*CDR*).

Call ID	Unique ID of a single event [*Integer*]
Subscriber	An active SIM card of a subscriber that is anonymized every day with hashing [*String*]
Network Identification	Subscring from the IMSI: “216” for Hungarian providers T-mobile, Telenor, UPC and Vodafone [*Integer*]
Device Type	Supplies the IMEI of each device anonymized [*String*]
Other subscriber	The other end of the telco exchange. Same rules apply as to the subscriber column.
TAC	TAC - ID of the exact type of the phone [*Integer*]
Type	Type of the telco event (1 - inbound call, 3- outgoing call, 7 - received SMS, 9 - sent SMS) [*Integer*]
Timestamp	Timestamp of the telco event in *YYYY-MM-DD**T**HH:MM:SS* format [*Timestamp*]
Cell ID	The hash of a telco tower - unique across all providers [*String*]
*Latitude**	Latitude of the tower [*Double*]
*Longitude**	Longitude of the tower [*Double*]

A few discoveries have been made while looking the real dataset and were concealed during the analysis of the sample data. Firstly, attempted calls and texts are being recorded regardless of the other subscriber’s status or response. Secondly, long text messages (multiple SMS) looks to be registered as individual telco events. This means their “call id” is different every time. It is more like a suspicion, than fact - it hasn't been confirmed. There are a lot of examples SMS going from a specific subscriber to another one within seconds between each other. While it is not my competence to decide how fast one can type an SMS, but it seems just a bit too fast. Thirdly, when a subscriber tries to connect with another subscriber really badly (rapid attempted calls and SMSs) - but the other one is unreachable for some reason - a lot of telco logs are being registered. This is problematic, because the crowd analyzers and predictors work with the count of telco data generated at a given area and time and further boosts the size of the log files. An important optimization step could have been to filter out these events - with the possible help of machine learning algorithms. Another optimization regarding source dataset size could have been to convert given *csv* formatted data into dataset that were created for big data analysis like *apache avro* file or *apache parquet* files. The common of these file formats is to have an additional metadata information about the schema and data type of the source files beforehand - we *do have* this information - to encode them in a binary format. This reduces the size of the files significantly and have a great advantage when cached into memory - something Apache Spark is famous of.

There was a last dataset available called the **NGPRS** dataset that contained a lot of extra and redundant informations: handovers, protocol switches, etc. The size of each file is ~16-20 GB and is far out of our resources to analyze them in a meaningful time frame. Of course it promises more accuracy and granular data, but for the predictors the MSC dataset is a perfect basis.

Now that we are familiar with the CDR data and its' magnitude let's leap into the framework which is responsible for all data processing and computation highlighting the aspects that makes it the perfect tool for structural big data analysis and the first steps (with Python) that made the transition to Spark necessary.

3. Data processing Framework

Keeping the magnitude of the data in mind and for the sake future scalability finding the right data processing framework was key to achieve the targeted objectives. Starting with the sampled data (~3 MB) the initial data cleaning, basic statistics and cellular tower location extraction were done locally in **Python**. It has great extension libraries in the form of *numpy*, *pandas* and *scikit-learn*, which are widely used in data analysis. Using it is really intuitive and possesses several analytical tools and machine learning algorithms. Kicking off with the first two analysis (how far people travel and find longest paths per day) Python quickly started to show performance weaknesses when scaling up data. The ease of usage made the starting to be quicker and we could get away with vertical scaling, but the objective is to be able to deploy the backend application to cheap, horizontally scalable clusters to be able to process as much data as possible. Python being a scripting language with no type safety or functional prowess rapidly lead to verbose, unmaintainable and unscalable source code. Older version of scikit-learn could not benefit from multiple cores, even so newer versions could utilize all cores memory and the absence of horizontal scaling led to look for something else.

Another solution had to be found. The criteria to the data processing framework is to be state-of-the-art, open-source, maintainable and has seamless horizontal scaling. The natural selection was **Apache Spark** [5] that had everything that was mentioned before. It is state-of-the-art, but being a couple of years old now mature enough to fill the role to be the engine of this whole project. Scaling and speed difference is significant when comparing to Python or even other big data ecosystems like Hadoop/Mapreduce.

3.1 Apache Spark

Apache Spark became the de facto big data tool to go nowadays. It promises lightning fast cluster computing and it surely delivers. It is performant, easy to use and still highly customizable and has couple of powerful libraries to facilitate data analysis and machine learning. Spark rapid succession lays in generality, ease of use, speed and wide range of deployability options:

(1) Run on Hadoop YARN, Apache Mesos or standalone cluster mode. HDFS file system is supported.

(2) Fast in-memory computing with intermediate result caching - this feature can be exploited in the source code giving developers more control.

(3) Multiple language API support for Scala, JAVA, Python and R. Scala was the chosen one here, because it has the most matured API out of all others, strongly typed and functional. Spark is also written in Scala itself and has a pretty nice dependency manager called SBT.

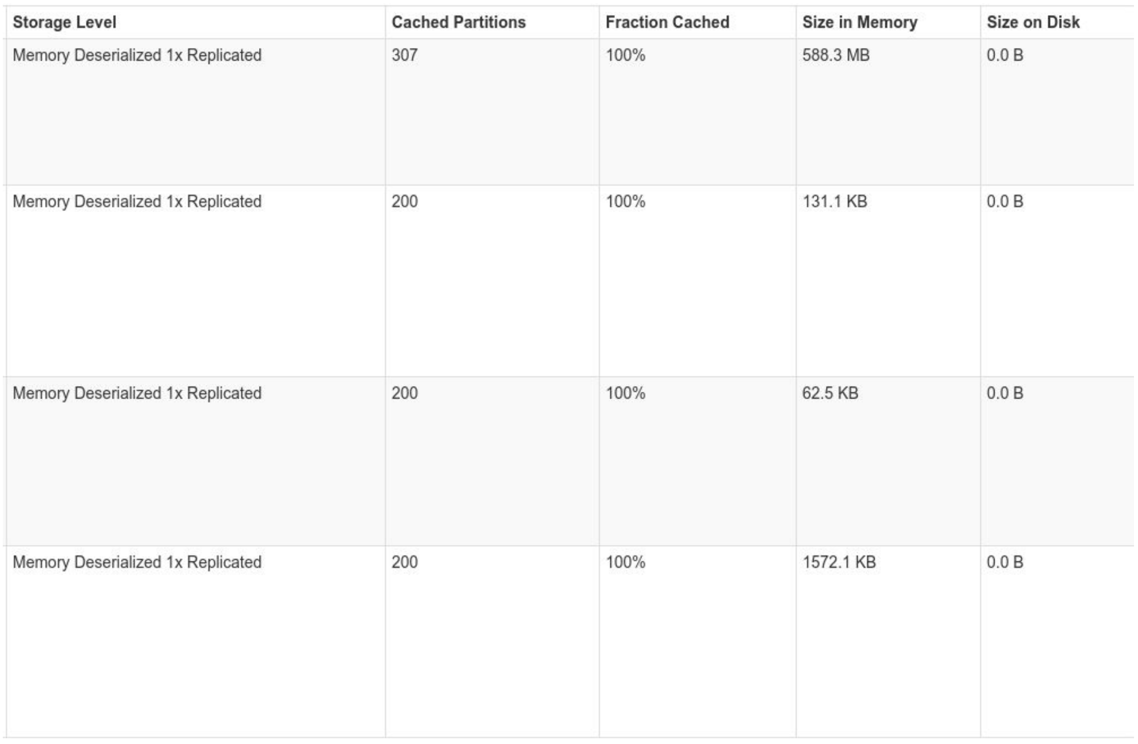
(4) Great package support: **Spark SQL** [6] and **Spark ML** [7] are center pieces of Spark big data environments with invaluable support for structural data, query optimization (Catalyst), execution engine optimization (Tungsten) and machine learning library.

With flexibility, performance, scalability and the needed machine learning algorithms everything was ready to shift the implementation from Python to Scala/Spark. In conventional big data processing the data is often unstructured and very diverse, but it is not the case here. We know the schema of the data in advance and as we have seen in the sample dataset there are little to no errors or outliers found in the initial set. Spark SQL is the perfect tool to handle structured big data that leverages the power of Catalyst [8] optimizer, which works with the newest Dataframes API. To put Catalyst into practical use as much as possible several Scala case classes were added to the code. **Pipelines** were constructed in a way that even intermediate steps conformed different predefined schematics. This technique helped preserving type safety in intermediate steps, caching and keep the code clean.

Spark leverages *resilient distributed datasets* (**RDD**s) that are fault-tolerant collection of elements that can easily be parallelized or cached into memory. Some intermediate steps were cached while constructing the datasets for the algorithms and that boosted the performance significantly. Datasets that will be reused later - and could fit into the memory - should be persisted into the cache. The cached datasets can be manually evicted when not needed anymore. There are *transformations* and *actions* on the datasets. Keep in mind that in this distributed environment with a functional language practically everything is immutable. Whenever a transformation takes place a new dataset is created. Mapping, filtering, grouping, joins, partitioning or sampling are transformations. They are *lazily evaluated* meaning they do not execute right away. The transformations are done when an action is called. Actions collect the distributed data together for further usage or to write out to a file. Reducing, collection, counting, saving to file or for-each are all actions. The actualized transformation and action lists can be found here: [10]. The lazy loading of transformations allows efficient execution and Catalyst optimizations to take place.

There are a lot of pitfalls here, so one should be aware of the underlying mechanism of Spark and distributed computing altogether. Mundane techniques such as for-loops should be avoided, unnecessary sorting or operations that could result in unneeded data moving around the cluster. In some cases Spark explicitly sends warn messages about possible performance degradation, memory leaks/spills or if an rdd has been cached already. These are usually programming errors and can be avoided, but sometimes exploited as well. I knew that I am going to run my Spark application on a single powerful machine, so in very rare cases have skipped data partitioning that would distribute partial datasets (resulting in data shuffling) around the cluster.

I might add that Apache Spark has an excellent web user interface to follow what is going - what tasks are being processed, their state, the datasets that were cached/persisted, number of executors and even SQL execution queries made by the Catalyst engine to mention a few. Time taken to execute each task can be tracked and if the execution is stuck the process can be killed remotely from the UI.

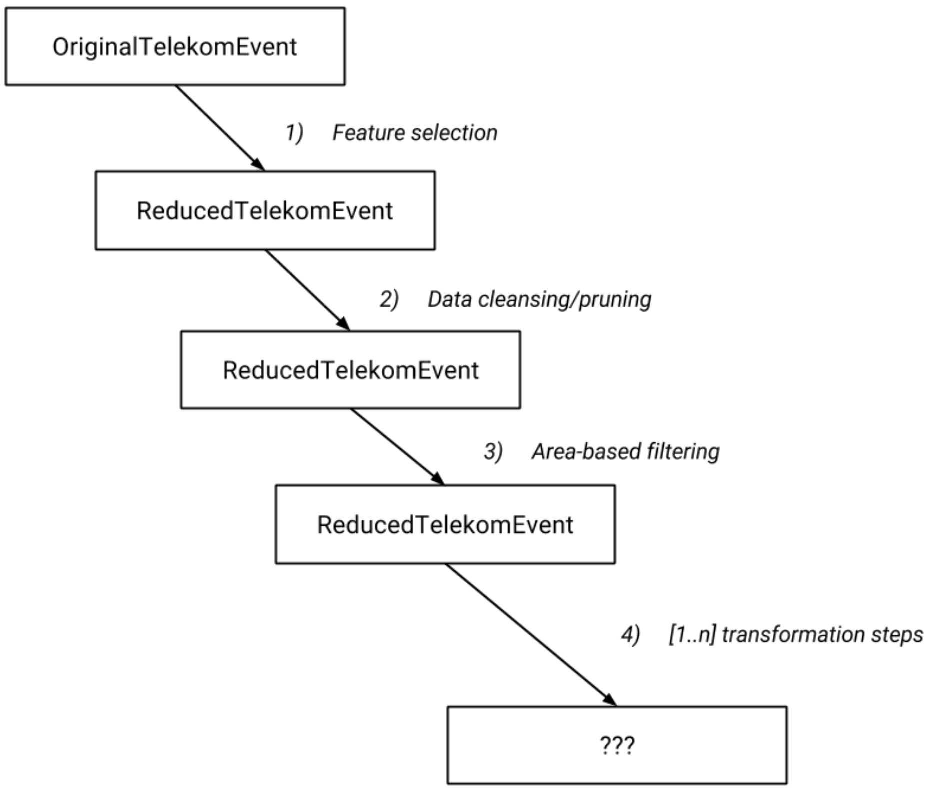


Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
Memory Deserialized 1x Replicated	307	100%	588.3 MB	0.0 B
Memory Deserialized 1x Replicated	200	100%	131.1 KB	0.0 B
Memory Deserialized 1x Replicated	200	100%	62.5 KB	0.0 B
Memory Deserialized 1x Replicated	200	100%	1572.1 KB	0.0 B

Apache Spark UI. The implicitly and explicitly cached RDDs in the memory - their size, percentage of readiness and the queries leading to them can all be tracked.

3.2 Pipelines

Transformations and actions create pipelines that are executed in order in a distributed fashion. Pipeline is not a Spark idiom (but will be introduced in Spark ML), just a phrase I prefer to use. The distinct tasks (distance evaluation, longest paths, crowd movement tracking and crowd prediction) were designed as stand alone, configurable jobs. However, there are common functions needed by all algorithms such as data cleansing or filtering. The order of these tasks are more critical than we might think. I have fallen to the trap to firstly filter out rows that contained invalid, null or missing values. Feature selection was the next major step and the realization came that I might have filtered out rows based on features I do not even need. Below a generic pipeline can be found I have came up with to kick off any analysis job.



Generic transformation flow of telco data. Some filter out rows while others change the schematic/columns.

The intuition is to start with the most generic filtering and advance with the more and more specialized ones. The labels inside the boxes are actual Scala classes that are used in the backend application. Let’s inspect the individual steps and the motivation behind them:

(1) **Feature selection** (*OriginalTelekomEvent* -> *ReducedTelekomEvent*): This transformation step alters the schema of the dataset to select the only needed informations: *subscriber*, *timestamp* and *latitude*, *longitude*, while creating a new column *dataset* from in MM.DD format from the timestamp. Vast majority of analyzers here focuses on daily data, so dedicating a column to it is a convenience especially in a sense that most grouping and partitioning happens based on this attribute parallelizing daily data processing over the cluster.

(2) **Data cleansing** and **data pruning** (*no schema change*): Having so few attributes left this step is rather easy. If a row has missing or null value in any of their attributes the entire row is discarded. The latitude and longitude is further validated, for example filter out (0, 0) coordinates.

(3) **Area-based filtering** (*no schema change*): Now this is something unique logic added lately to the pipeline. In the beginning data from all around the country is at our disposal. If we want to study specific focus areas then we want to filter out every tower that is not in our range of interest. There are three specific areas that were defined here:
- *Outer Budapest*: ~20 km circle around the capital with the suburbs and M0 freeway included
- *Inner Budapest*: The heart of the city with a 2,5 km radius circle including most tourist attractions. The density of the telco towers are the highest here.
- *Ferihegy Airport*: A 1 km radius circle around Terminal B with just a few towers.

(4) **Other transformations** (*ReducedTelekomEvent* -> ***): Here the task specific transformations happen. These specialized pipelines will be highlighted in the discussion of the specific tasks in details.

The area-based filtering is of course optional - it is disabled when working with nation-wide data. More area filters are easy to add to the backend framework: one just needs to specify a center point (latitude, longitude) and a radius in km.

3.3 Ecosystem and environments

A list of backend modules and their respective versions can be found in the table below. Every module's and package's latest *stable* version was put to use.

Module or package	**Version**
SBT	0.13.13
JAVA	1.8
Scala	2.1.18
Spark Core	2.1.1
Spark SQL	2.1.1
Spark ML	2.1.1

The SBT package- and dependency manager built for Scala made setting up the local development environment a breeze. On the not so sturdy local development environment (Hp Pavilion 15 notebook - core i5, 4GB RAM) there was no Spark deployment. With a few lines of code in *build.sbt* file the given dependencies were downloaded, loaded and started on demand when the application started up from the SBT console or the IDE.

JAVA had to be installed mostly because of the JVM it provides for Scala to run in, but Java core packages and classes can be used in Scala context out of the box, which came in very handy from time to time. Scala was needed of course, because Spark was written and Scala and so does the backend processing application. Scala is the only exception where not the most latest version is used simply because the proviso of Spark using a bit older version.

Spark Core package provides the foundation to big data analysis, Spark SQL and Spark ML are only extension libraries the former providing SQL-like functionalities, Dataset API and optimizations (Catalyst, Tungsten) for structured data and the latter giving powerful machine learning capabilities. Their versions are synchronized and it is a good idea to not mix up their respective versions.

The actual production environment consisted of a single, but powerful academic computer. This was unfortunate in a sense that the project was built up from the ground keeping distributed cluster computing in mind. Still the implementation could take advantage of parallelized computing and execution over the cores and threads. Horizontal scaling of resources and the scaling up of the size of the data should require no additional tuning. The production computer had 2 x Intel Xeon-E5540 processor (4 core / 8 threads) with 16 threads available altogether, 64GB memory and SAS storage and Ubuntu Linux. Production environment had JAVA, Scala and Spark preinstalled. The application’s fat jar had to be uploaded and could be run from the command line interface directly.

4. Visualization Framework

Having the ability to see the results of the backend processes helps understanding them greatly. While the primary focus point of this thesis is backend processing, an aesthetic visualization makes the whole application a true framework. As opposed to traditional visualization we have an added complexity to deal with the massive amount of output data that could come from the backend system. The visualization was implemented as a web application with the aid of HTML5, CSS, various open-source javascript libraries and the Google Maps Javascript API [9]. The Google Maps API needed explicit registration and a unique API key is provided in return that is needed to query against it. There is a 25.000 map load per day limit as far as the product is not for commercial use.

A simple Python webserver serves as the backend for the data visualization. Data loading for the frontend are asynchronous calls to local files not to block the user interface (UI) while new data is being pulled for the visualization framework. Spark constructs the output files with such a layout that these files can be loaded right into the visualization framework without any further refinement. This made changing view between the different sub results (e.g. outer Budapest vs inner Budapest or sample dataset vs result dataset) very convenient.

Summary of the used javascript libraries.

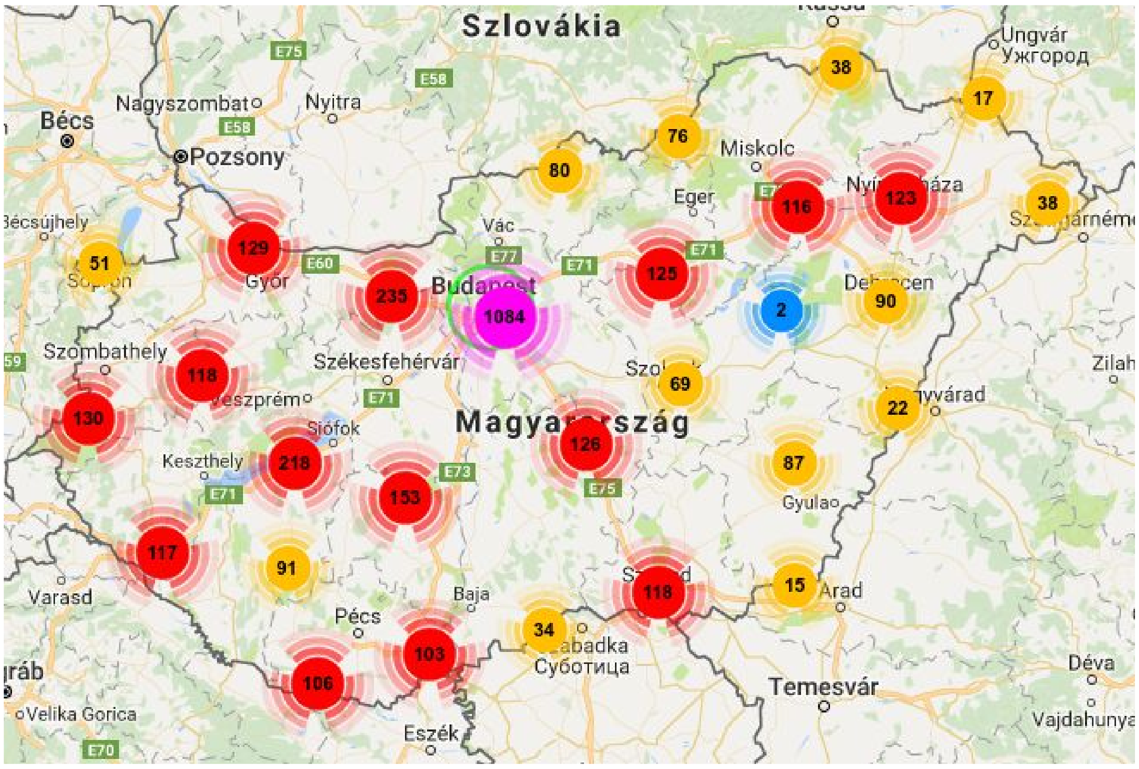
Library	**Version**
jQuery	3.2.0
Bootstrap	3.3.7
Google Maps Javascript API	3.27
D3.js	4.8.0
Google Charts	45
jQuery UI	1.12.1

Without diving into too much details regarding the mentioned libraries I just give a quick glance on their use cases:

- *jQuery* and *jQuery UI*: DOM manipulation and user interactions
- *Bootstrap*: layout and responsive view support
- Google Maps JS API: map, markers, lines, heatmap and automatic visual marker clustering support
- D3.js: file loading and frontend data transformation logic
- Google Charts: charting library for statistical data visuals like histograms, box plots, etc.

The responsive design eases the use of the front interface from various devices, screen size and resolution, moreover the fluid layout automatically adapts to fill the screen with content. The frontend part is a separate project from the backend part and could work well with other data sources as long as these sources conform the currently used data schematics. If the backend driver program and the visualization program would run on the same machine - with the proper configurations done - new results from the backend processing could be made available in the frontend instantly giving a pseudo live streaming feel.

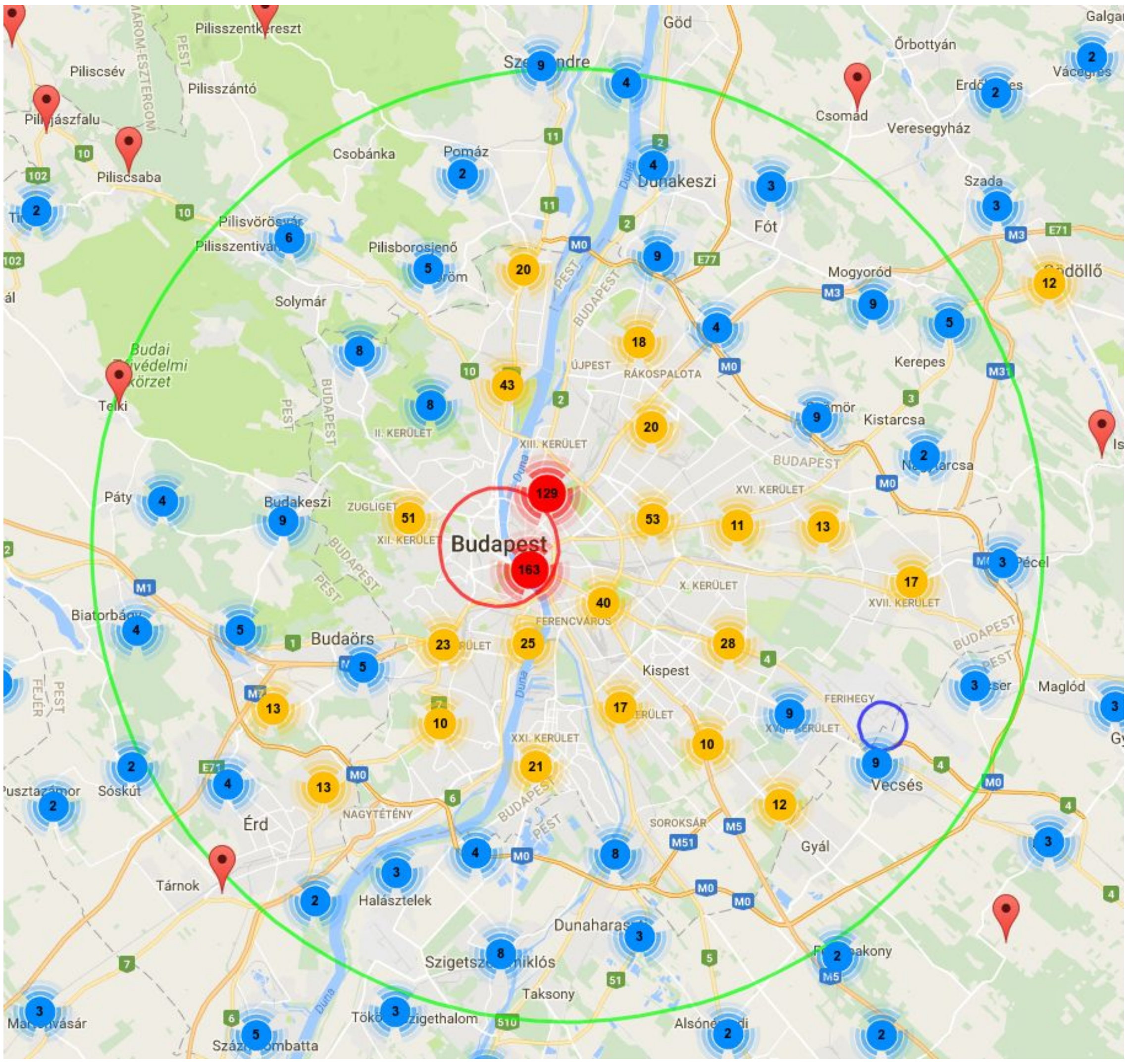
If anything in this framework the frontend part could use a major revamp. Data loading currently is limited to loading data files with predefined names and schema to display therefore flexibility is lacking. Anyhow, the main attraction of this thesis is the backend analysis and the performant parallel computing, the frontend is merely limited to better showcase the achievements.



Cellular towers clustered over Hungary.

In order to get a bird’s eye view of the cellular network in the country the towers were visualized. Since exact cellular tower locations are known they can be placed directly on the map with the help of map markers and in addition automatic clustering can be toggled on for easier distant views. With zooming in or out on the map the clusters are recalculated and if the towers are granular enough then exact tower locations become visible.

Tower placement gives a basic idea about population density and crowd at a given place in any times. Zooming in to Budapest the hearth of the city is is easy to be seen - even if we have no idea about the city' borders - and also marks the suburbs around the capital. In the center there are towers at just about every other street while in rural areas a single tower may cover a few km^2 alone even up to 10+ km^2 supercells.



Cellular towers clustered over Budapest.The circles mark the focus areas: (1) green circle: Budapest and suburbs, (2) red circle: Budapest downtown area and (3) purple circle: Ferihegy Airport Terminal B.

5. Distance analysis

In order to have a basic understanding of mass mobility the first metrics that was analysed is the distance people travel each day. More travelling activity could indicate more crowd on the routes at the given day. I have a preconception that the two most busy days are Fridays and Sundays for obvious reasons. Can this suspicion be validated from the distance analysis? How external effects (e. g. extreme weather conditions) have an impact on people's travel distances? Is the method presented here even suitable for such task? How does the findings gained here are in line with the output of other analysers? There are a lot of questions to answer and a lot more could be asked.

The main idea was borrowed from [1], but have been taken further. The daily travel distance of a single person is the maximum distance between each cell tower the user's phone came in contact with during the day. There are problems with this calculation: CDR inaccuracy has been shown already, but give or take a few kms is not a big deal after all. The bigger problem is that we have no data from every area each user visits. People tend to initiate calls during travelling somewhere to save time. In addition on days when more telco exchanges happen we might have higher numbers - simply because the higher probability that the user reaches out to an edge tower during travelling. Remember that CDR data is only collected upon initiation or receiving of a telco exchange. During the analysis the data is partitioned per day and grouped per user. The individual results are aggregated together before being written out to the disk. An indirect benefit of this approach is to preserve privacy protection and make the data more lightweight for the frontend system. This approach is used in every other analyzer as well the only exception being the longest paths analyzer, where individual paths are being examined.

The following metrics were extracted from the telco dataset:

(1) Count of telco events per day

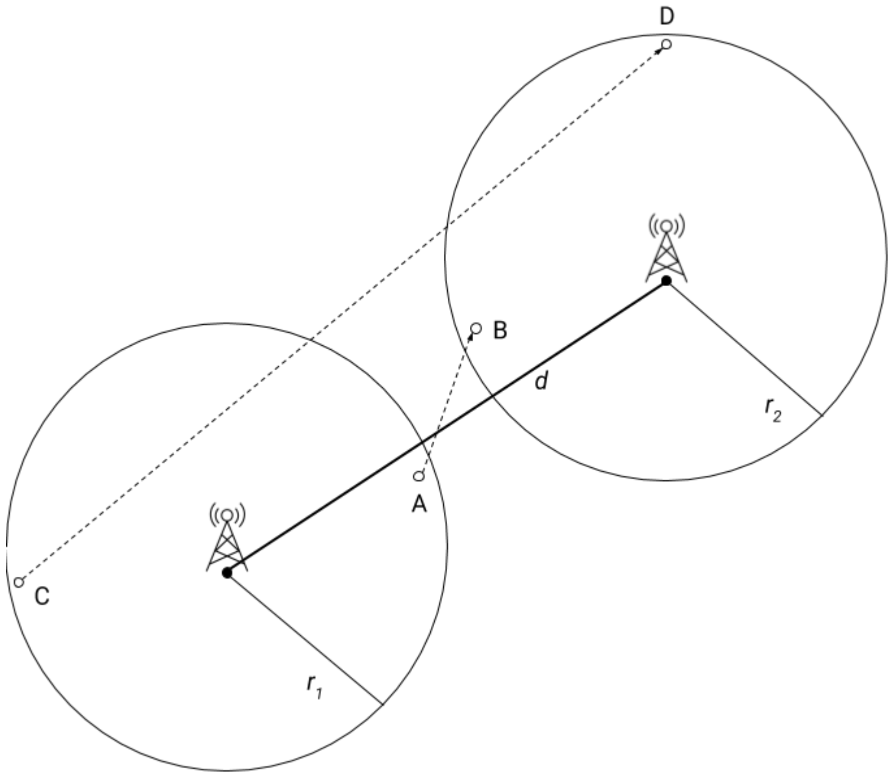
(2) Maximum distance covered per day

(3) Quantiles per day

(4) CDF distribution of the distances per day

The count of telco event is indirectly related to distances only. Higher count means more cellular activity and possibly provides more fine-grained distance information. In the sample set latitude and longitude values are known.

Another thing to consider is how accurate these distance values are? The distances calculated with Haversine formula (next section) introduce negligible error. Recall what we have seen previously - the CDR inaccuracy (*section 2.1*).



Given a mobile user to travel between two cells. The recorded distance is *d*, while the actual distance could be A-B or C-D distance.

The image shows two non-intersecting cells with radius r_1 and r_2. For instance a mobile user travel from one to the other. If the user travels from A to B or C to D the recorded distance is *d* anyways and does not reflect actual distance covered. The actual distance is bounded between [d-r_1-r_2; d+r_1+r_2]. The truth is that these distances "balance out" over big quantities of telco data. The data could be skewed with small portion of telco data, but that's why big data analysis is here to save the day.

Fluctuation between weekdays and weekends is something to keep an eye on. The initial charts below are based on nation-wide cellular activity. Wherever a considerable amount of divergence comes forth from these patterns while analysing with area-based filtering it will be discussed, otherwise not.

The real dataset spans from 15th of September (Thursday) to 28th of September (Wednesday), 2016. Fridays (16.09., 23.09.) and weekends (17.09.-18.09. and 24.09.-25.09) are days to look at since other weekdays show very identical results.

5.1 Haversine distance

The distances between these pairs calculated using the **Haversine formula** [11]. The haversine distance suffers from a tiny error margin due to the fact that Earth is not a perfect sphere, but a *spheroid*. Given two latitude-longitude pairs it calculates the great-circle distance between them. This distance is the shortest path's distance over the surface - it ignores geographical features (like lakes, mountains, ...) or routes.

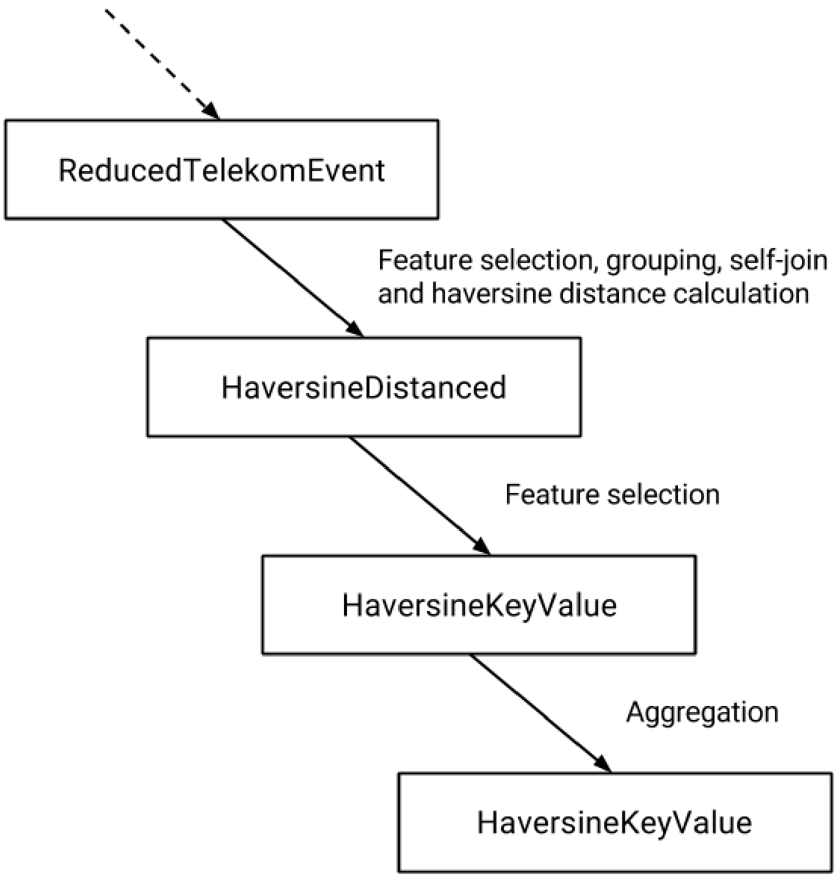
The Haversine formula that had been implemented [12]:

$$a = \sin^2(\Delta\phi/2) + \cos\phi_1 \cdot \cos\phi_2 \cdot \sin^2(\Delta\lambda/2)$$
$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$
$$d = R \cdot c$$

where ϕ is latitude, λ is longitude, R is earth’s mean radius and the angles are in radian. To calculate in km use R=6371 or R=3956 to calculate in miles. For the calculations the former value was used.

5.2 Distance analyzer pipeline

To conduct the distance analysis the data have to be partitioned by day and then by subscriber. Secondly, every cell’s distance have to be calculated from all other cells the subscriber visited that day. After finding the maximum of these the data have to be collected together as *HaversineDistanced* class records that holds dataset (month and day), subscriber and distance columns. We do not care about individual distances here so data is further aggregated by the dataset attribute to reach the *HaversineKeyValue* class that holds dataset as key and the values are various aggregations of distance data (min, max, quantiles, ...) dropping subscriber column completely. This dataset was the base for every analyzer jobs here, so I have explicitly made Spark cache it for faster execution. The intermediate *HaversineDistanced* class is needed, because the next analyzer (longest paths) will take advantage of it, gives more supervision over column types (Spark throws exception in case of a type mismatch), makes the code more readable and allows Catalyst optimizations.



Data transformation flow for distance analysis.

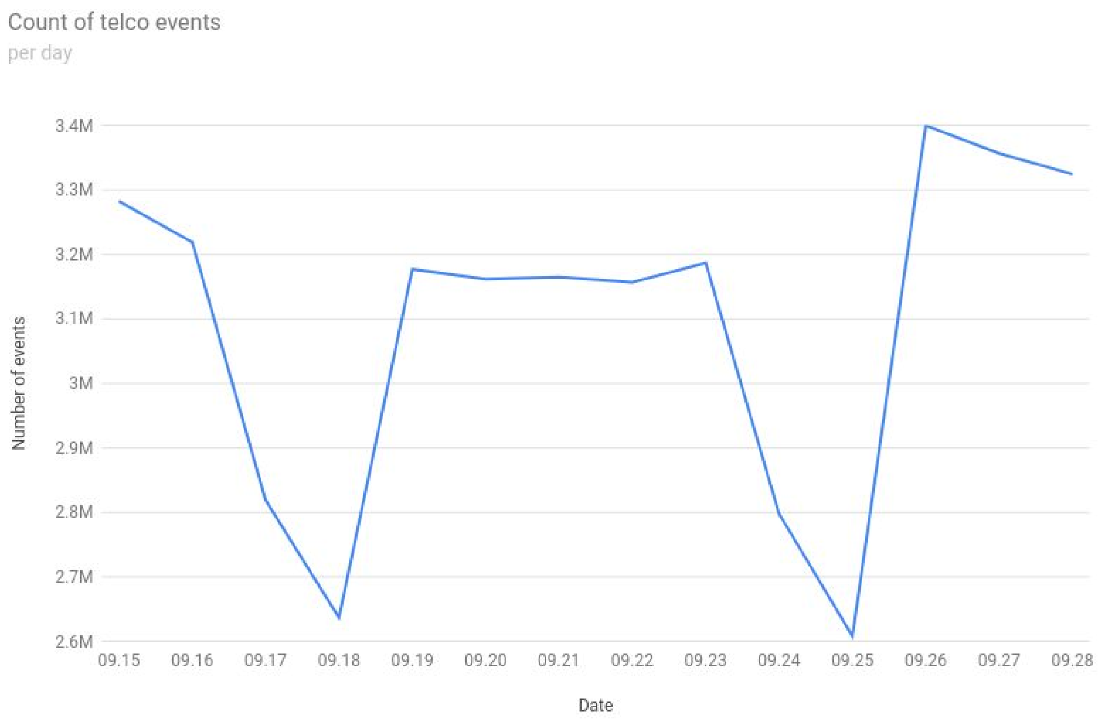
HaversineDistanced class.

dataset	Date information in *MM.DD* format [*String*]
subscriber	An active SIM card of a subscriber that is anonymized every day with hashing [*String*]
haversine	Maximal distance the subscriber travelled [*Double*]

HaversineKeyValue class (final step).

dataset	Date information in *MM.DD* format [*String*]
haversine	Aggregated distance value (min, max, quantile, ...) [*Double*]

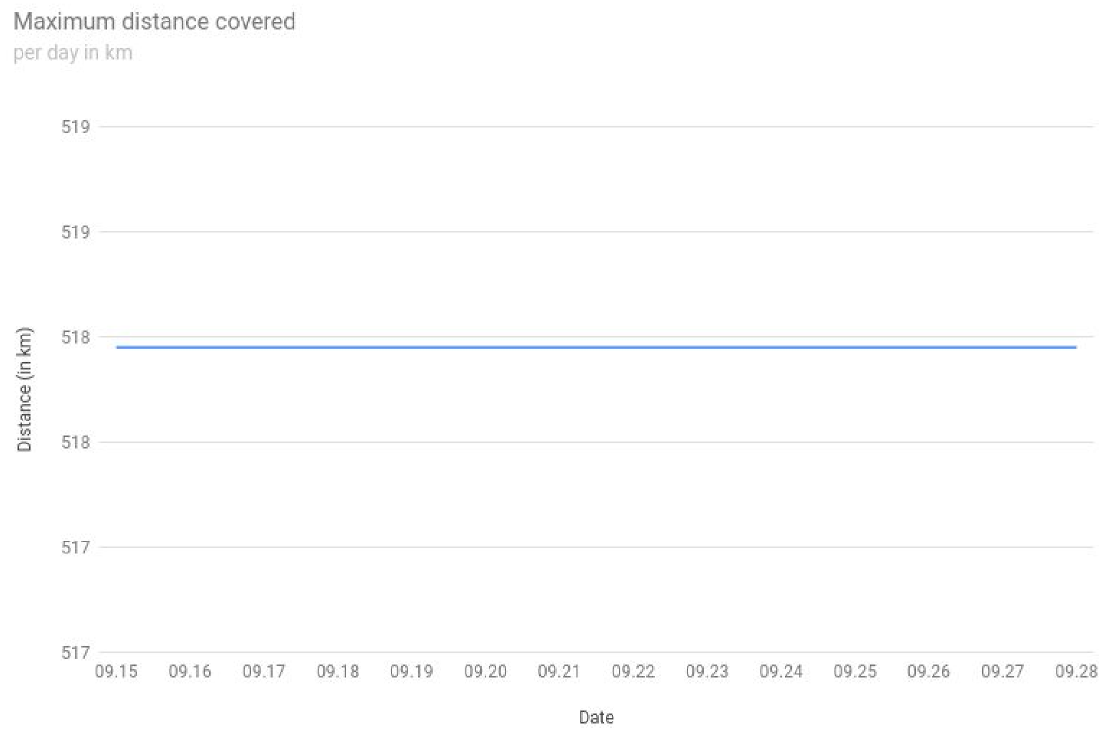
5.3 Event count analysis



The y-axis shows the number of individual telco events and the x-axis shows the days.

As we can see the weekdays are pretty evenly distributed with the apex being on Monday (25.09.). The weekends' downfall are huge producing almost identical values over the two weeks with Sunday being on the bottom. Values of the Sunday on the 25th is a little bit below of the previous Sunday's values on the 18th. On the other hand the rise in the number of telco events on the upcoming Monday is far higher. Is there a relation between Sunday's values and Monday's values? A bigger sample set would be needed to arrive to such conclusions. Limiting the analysis to Budapest only the line is identical to this. Next let's analyze the maximum distances recorded.

5.4 Maximum distance analysis

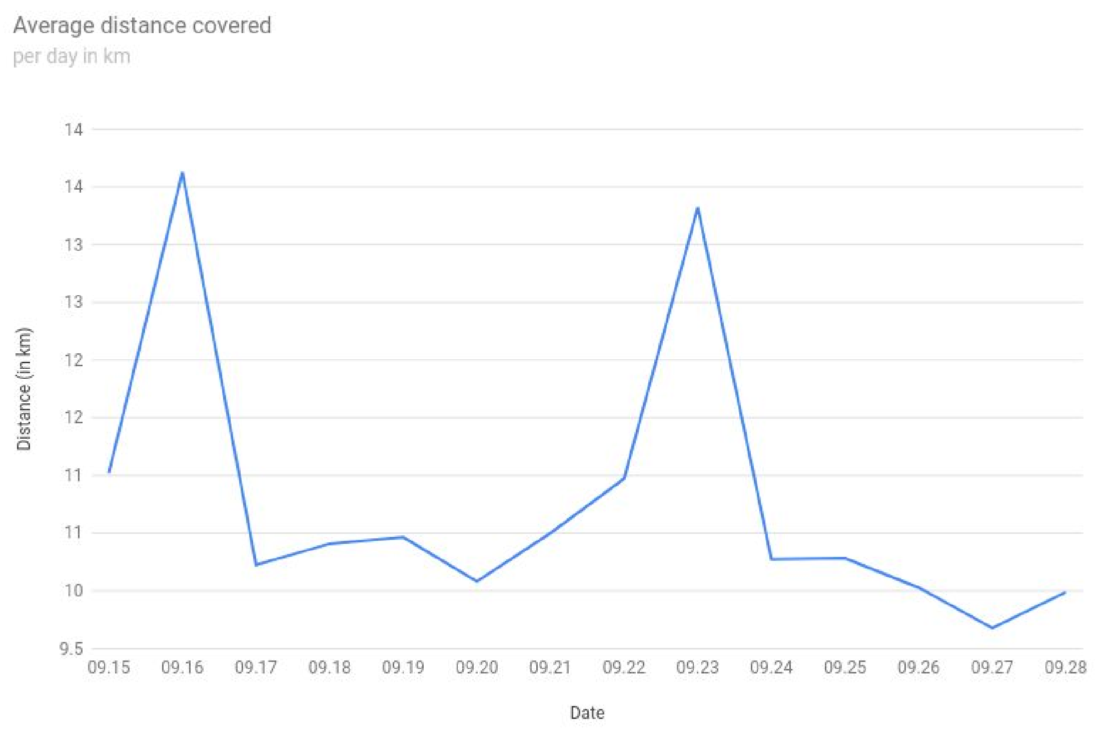


The y-axis shows the maximal distance recorded and the x-axis shows the days.

What kind of consequences can we interpret from this data? Not much, actually. Maximal distances were expected to be more or less the same on every day. In the sample dataset there is a 100+ km distance difference between days that must be the effect of the heavy sampling. With the real dataset we have a nice flat line with a constant 518 km values - which is believed to be the distance between the two furthers towers in the country. Narrowing it down to outer Budapest area we see a flat line of 39 km. Given the radius for that area is 20 km this value is completely valid. Same with downtown area: flat 4,9 km distance in a 2,5 km radius.

Maximal distances could potentially show outliers or anomalies with extraordinarily high values. The peak distance is exactly 518 km here - that is an entirely possible distance to travel within one day. If we check out the timestamps of the locations - with the haversine distance - we could calculate one’s velocity. That is an entirely different analytic job [2] and out of the scope of this thesis. Let’s dive into a little bit more descriptive metrics and examine the average distance covered per day.

5.5 Average distance analysis

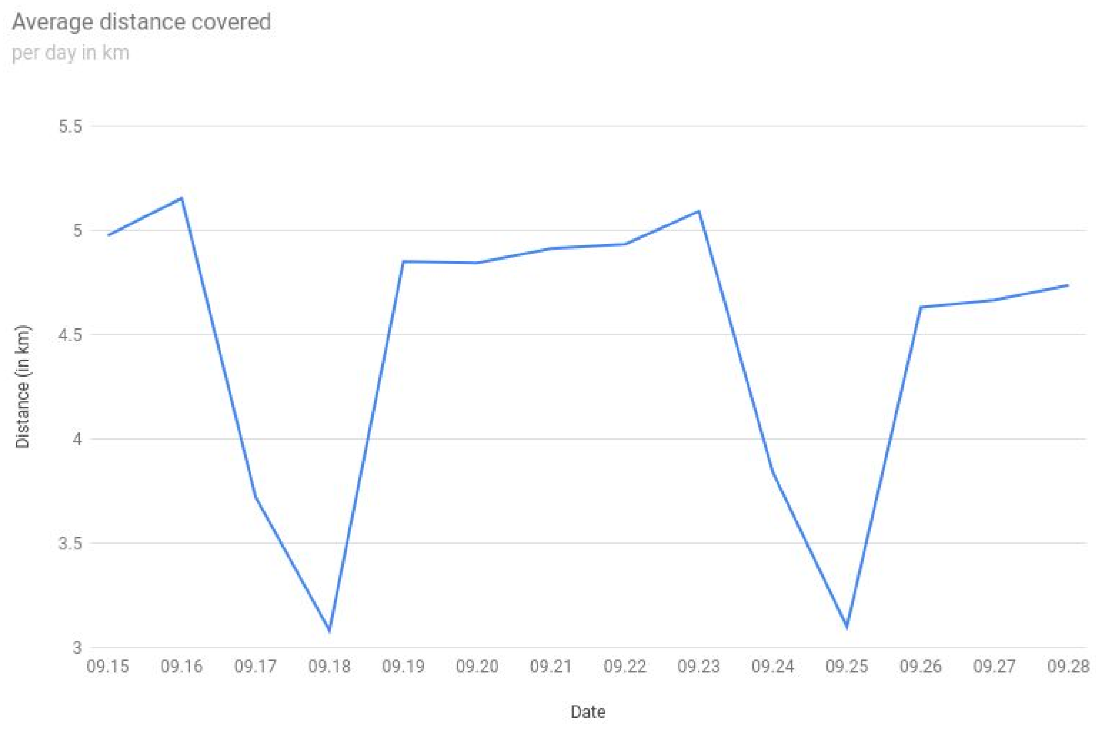


The y-axis shows the average distances and the x-axis shows the days.

Now that is something interesting. Fridays showing strength peaking far above the other days - this is when people go back home or to family from big cities to the countryside or smaller villages. Sundays - when these masses travel back - do not show any peak values. They do travel back of course - just do not use their cell phone during the weekends most probably. Tuesdays (20.09.,27.09.) are producing the lowest results, if one want to travel with the least activity pick

Tuesday to do so. Thursdays on the other hand are the second most busy days during the week. Weekends and other weekdays producing surprisingly similar results. To continue our research reveal the quantiles.

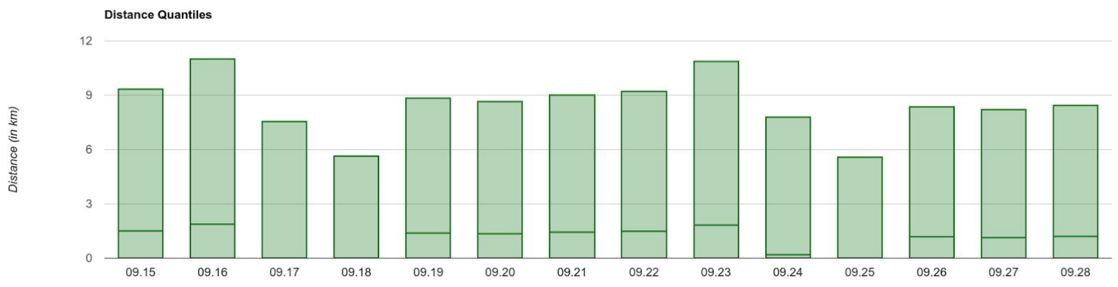
If I restrain the analysis to Budapest only we see a different result - something much more similar to the count curve.



The y-axis shows the average distances and the x-axis shows the days - outer Budapest area only.

This makes me conclude traffic/movement is more active outside of the capital on Fridays and going around in the city on Friday is not worse than any other weekday. Filtering to downtown area a similar curve emerges.

5.6 Quantile analysis



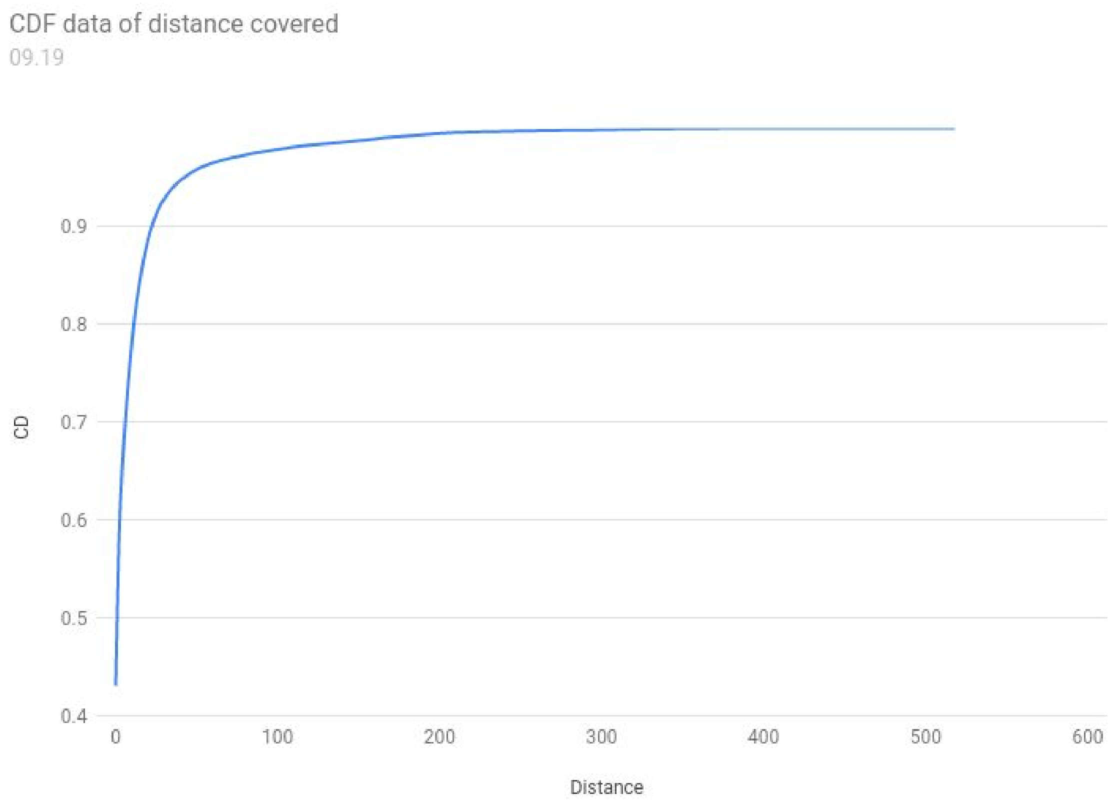
The y-axis shows the 0.25, 0.5 (*Median*) and 0.75 quantiles and the x-axis shows the days.

Fridays peak once again, while other weekdays score literally identical quantiles. What is interesting and could not be seen from the average chart is that the median is zero on the weekends, but not on the weekdays. This means that at least half of cellular users did not move during the weekends or was involved in one telco exchange only. Tuesdays did not produce any peculiar values - something has been seen in the average distance values. Bounding the analysis to outer Budapest and Budapest downtown areas shows results pretty much indistinguishable - with lower numbers of course. The only difference is that values from Monday's are slightly underperforming.

5.7 CDF analysis

Lastly cumulative distribution functions (**CDF**s) were drawn for each day. For this real valued distances were rounded to the closest integers according to standard rounding rules. The CDF is the probability that a random variable X takes a value less or equal to x. Formally:

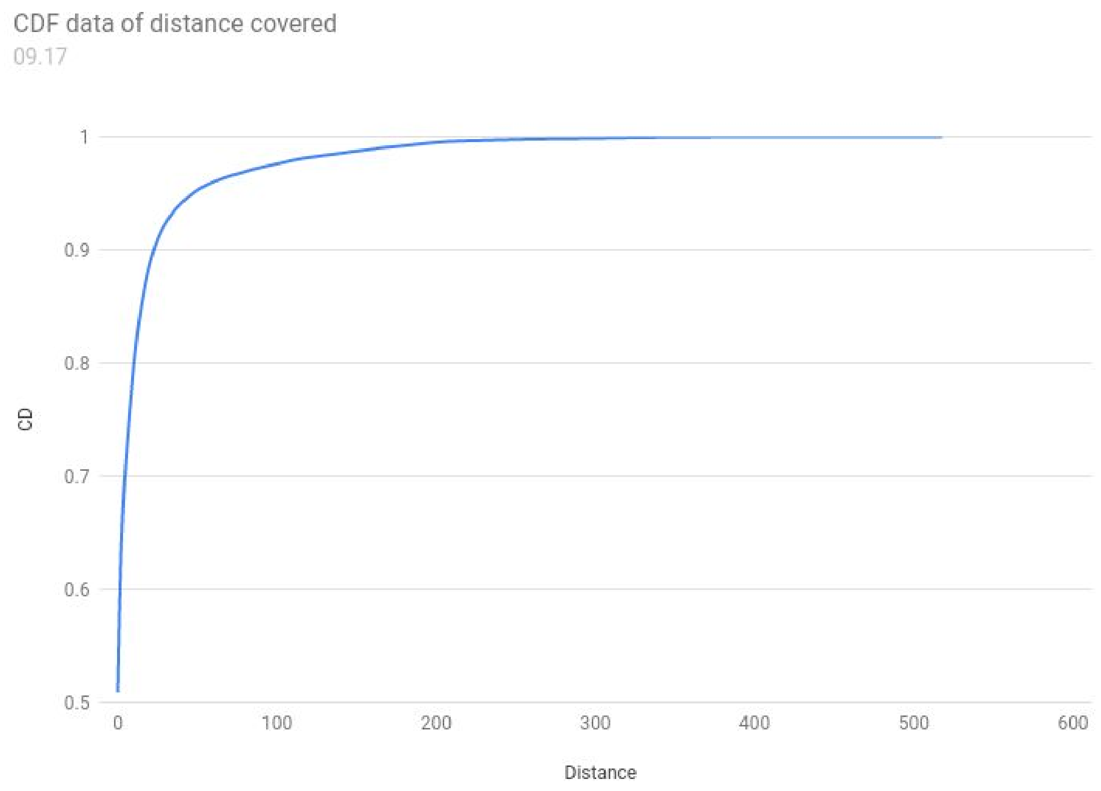
$$F_X(x) = P(X \leq x).$$



The y-axis shows the CD value and the x-axis shows the rounded distance values of a Monday (19.09.).

The line chart above shows an impressively steep CDF. 43 percent of cellular users did not even show movement on this particular day all around the country and only 10 percent travelled more than 22 km. The 99th percentile is reached at 137 km.

In contrast let's see a saturday's (17.09.) CDF - is it really different? Seeing the previous analyses we could have had arrived to this premature assumption.



The y-axis shows the CD value and the x-axis shows the rounded distance values of the previous Saturday (17.09.).

The CDF is very analogous to what we have seen. This time 51 percent of cellular users did not show any mobility (+8%). Only 10% of the users travelled more than 22 km (same) and the 99 percentile is reached at 139 km (+2 km).

We did not see those convincing differences between weekends and weekdays as we did in the other analyses.

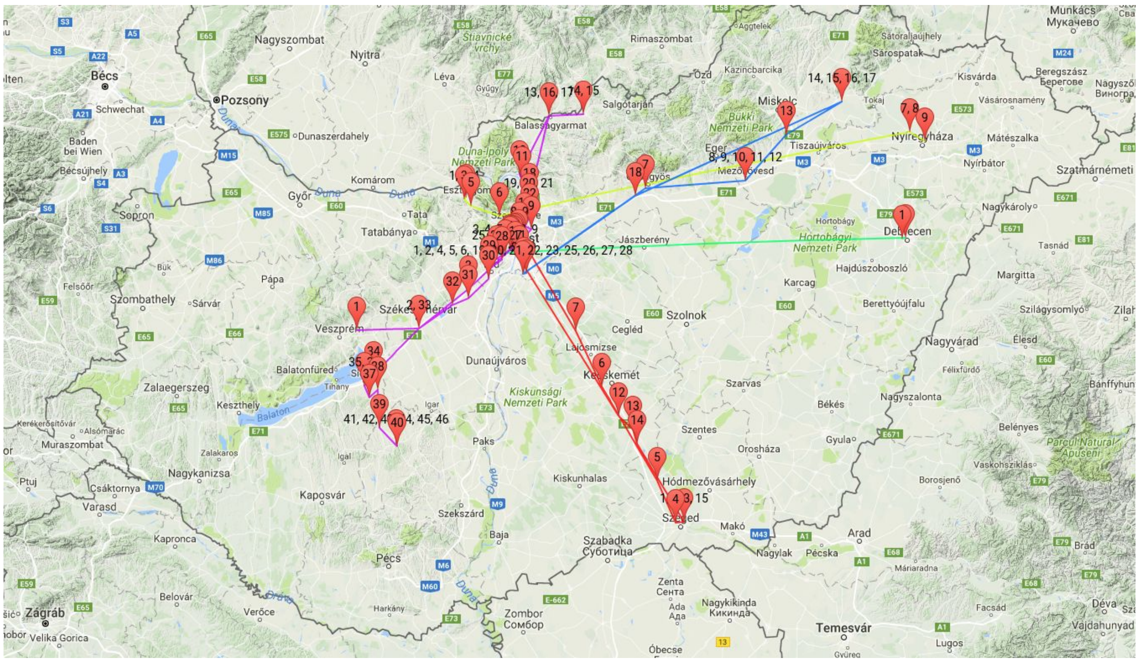
6. Longest paths analysis

The aim of Big Data analysis is often to get a big picture of something through data grouping and aggregations. The distance analysis described beforehand served a similar purpose. Another application is to find edge cases, outliers or anomalies. The longest path analyser exposed here falls into the latter use case. Similarly to the maximum distance metrics already shown here we look for the *k* longest paths done per day. This time actual distance does not matter, but individual paths (every cell on the path) will be visualized and looked more closely. In many cases it is easier to understand what is happening under the hood through actual examples rather than theorizations and assumptions. A striking reveal shows how our distance analyzer is more unstable than we previously thought it is and distorts our results. Before diving into actual examples my preconception was that longest paths will show traffic going on the hungarian motorway system (M0-M7). This held true for most cases, but there were exceptions.

The implementation used the *HaversineDistanced* class from the previous chapter. As easy at it seems to select the *k* longest distances it is not a straightforward task in a distributed environment. To achieve maximum performance one have to find a solution that results in the least data shuffling, easy to parallelize, contains no useless grouping or sorting. After having *HaversineDistanced* data (available from the previous computation) that contains (dataset, subscriber, haverine_distance) rows my approach was to send each daily data to different workers then order by haversine_distance (descending), rank them with a helper column than keep anything with rank ≤ k, then merge the results.

The problem with this dataset at the end of several transformations is that geospatial (latitude, longitude) and time details were lost long ago in the transformation process. In this immutable and distributed system there is no turning back. We have to filter the original (*ReducedTelekomEvent*) dataset with the newly calculated dataset and select rows with matching *dataset* and *subscriber* values. Thankfully the dataset holding the rankings contains only *d* * *k* rows, where *d* is the number of days and *k* is the limit to the number of longest paths we want to get back. *k* being relatively small (in this analysis *k*=5) this dataset could easily be cached into memory.

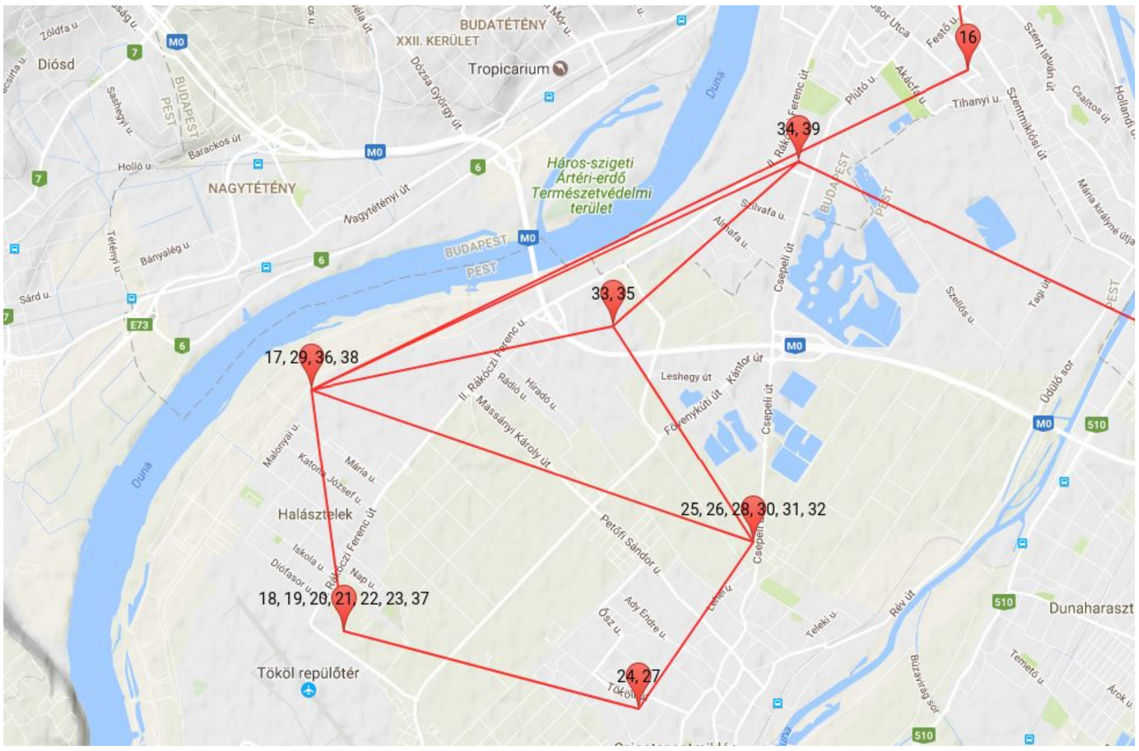
Finished with the theorization process let’s see some practical examples. The markers are numbered based on the order the user “visited” these places.



Each individual’s path is drawn with a different colour and the cells they came in contact with according to CDR data (October 5th, Wednesday).

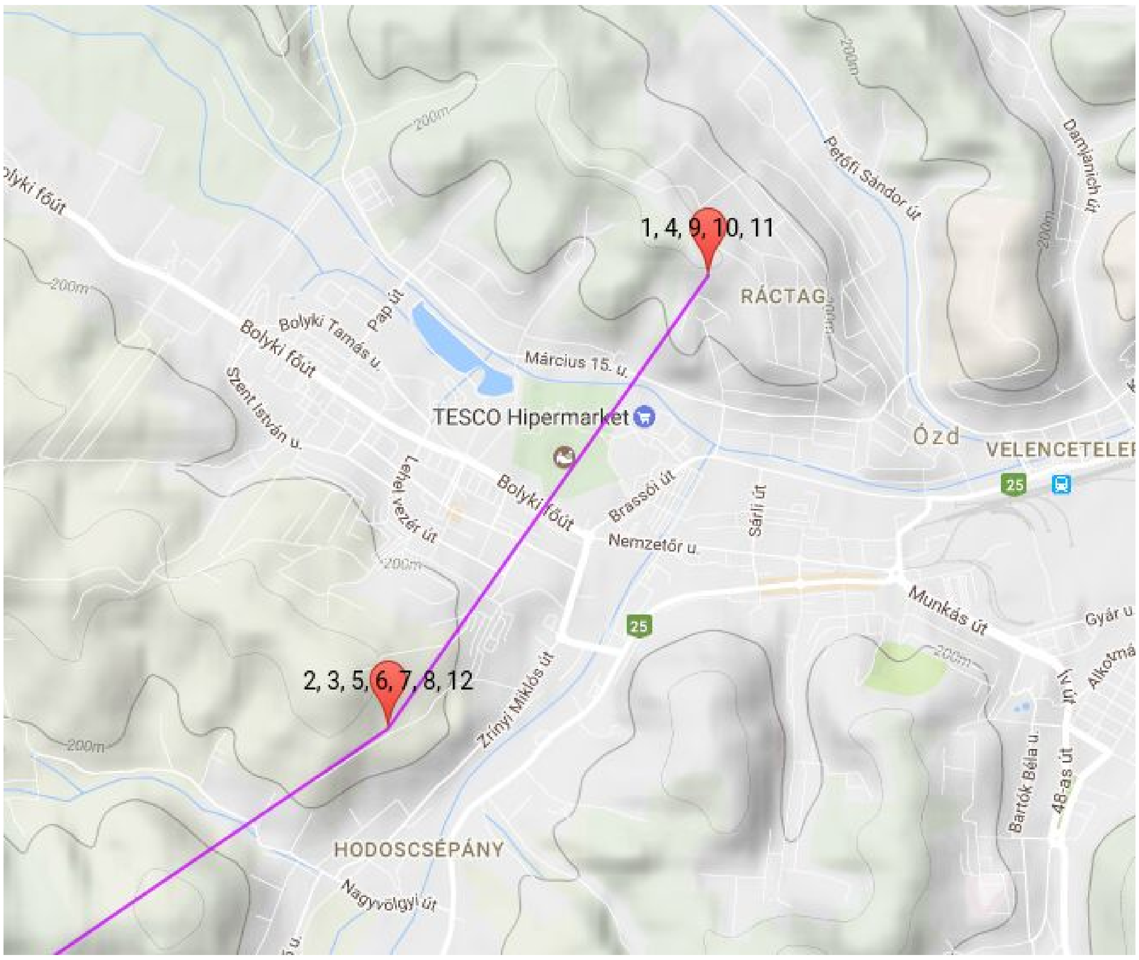
The pattern that spans between days is that people who travel the longest distances during their journey sooner or later come in contact with the capital, Budapest. Usually they start their journey from the countryside going to the city then go home or continue to the other half of the country. There are very few exceptions to this, but it shows how our usual paths centered around the capital.

What is much more interesting is how people *cheat* to score higher distance results! The “cheating” is not intentional of course and the source is the way phones select which cellular tower to connect to depends on a lot of factors. Let’s give a look to a specific subpath of the winner of the first day (October 3rd, Monday).



Our subscriber was very active just south-west of Csepel, Budapest. An alternative - and more likely - explanation is that he/she happened to be in the intersection of these tower’s radius rapidly changing which tower to connect to.

The given example below could be investigated with time analysis, but that is out of the scope of this thesis. There may be numerous cases like this: a subscriber sitting in one place exchanging calls and SMSs in two or more tower’s intersecting radius. What we see from the data is that the user is travelling, but it is not the case. Let’s look at another day, another subscriber at another place.



A small village with two cellular towers on the nearby hills. It depends on a lot of ever-changing factors which one your phone is connected to.

From what we have seen so far I would not say CDR data is an accurate predictor of distances travelled - not in it's current form at least. The data is really far-fetched and only gives a bird's eye view about what is going on. With a lot of fine-tuning much more accurate models could have been constructed.

7. Crowd analysis

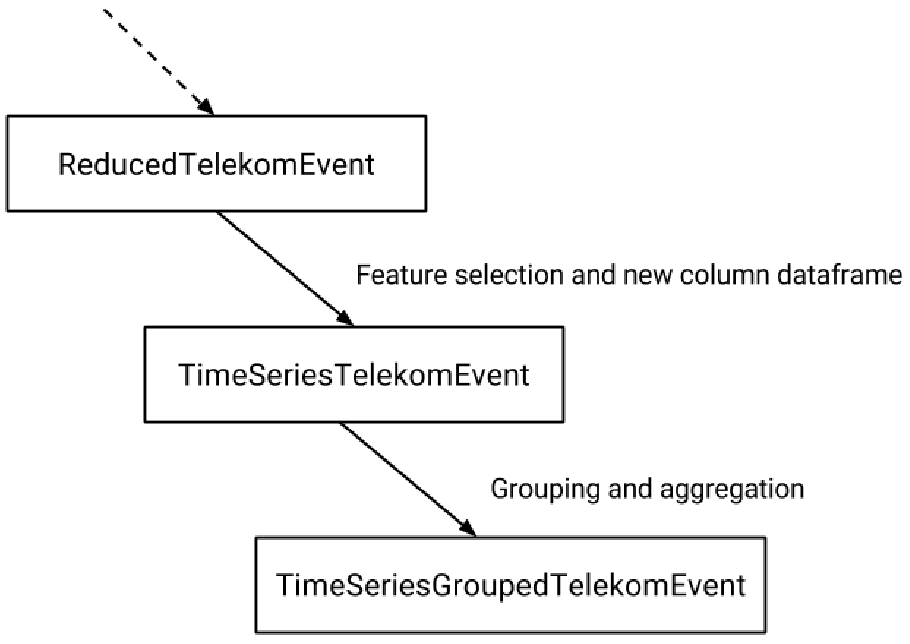
We have arrived to the point that is the focal point of this thesis: crowd analysis and crowd prediction. Before the predictors can be used a model has to be constructed that will determine how crowd is defined in a certain area. The definition is intuitive: the number of registered telecommunication events for a given cell will define the crowd in the area. Following the technique - which was used so far - that we work with daily data would not be granular enough here. The data was ordered by timestamp and discretized into time frames, thus essentially converting the dataset into *time series data*. The data was sliced with hourly precision resulting in 24 time frames per day, but a more granular grouping also makes sense to get more accuracy especially in busy hours like morning rush hours. Based on this time series data future values may be predicted being it next hour, next day or even further.

The substantial information here is to compare these absolute time series data with each other. Inspect morning rush hours on weekdays in relation of different parts of the city, how long do parties last during the night on each day and district, etc. The possibilities are endless here. There will be a few findings highlighted here and with the help of the visualization program that leverages Google Maps heatmap an interactive map is ready to begin experimenting with. A slider helps to switch between the different dataframes on a particular day. This chapter focuses solely on known or historical data analysis and visualization, while providing the starting point into time series regression analysis and classification (next chapter).

Visualizing the sample dataset once again proved to have limited usage and had only enough telco information about the capital. Analyzing the real dataset a much wider and deeper insight could be gained. Before starting in on actual results check out the pipeline of the crowd analyzer.

7.1 Crowd analyser pipeline

The new column here is named dataframe. It is an integer value mapped from the timestamp column that is known already. This made the schema to change, so a new class named *TimeSeriesTelekomEvent* had to be introduced. The mapping function is relatively straightforward here due to the hourly precision the hour is just extracted as the new column. The great thing about this distinct granular step is that the mapping function could be replaced easily. If we want time frames to be half an hour precise or even five minutes precise that can be achieved with minimal effort. A more creative thing would be variable-length time frames: more precision during morning and evening rush hours (e.g. between 5am-9am and 4pm-6 pm) and less precision during the day. It is up to our computing power, data availability and point of interest how we fine-tune this feature.


ReducedTelekomEvent
Feature selection and new column dataframe
TimeSeriesTelekomEvent
Grouping and aggregation
TimeSeriesGroupedTelekomEvent

Data transformation flow for crowd analysis.

The *TimeSeriesTelekomEvent* rows had to be grouped by dataset, dataframe and cell and a simple aggregation (count) returned the desired crowd numbers. This final class is the *TimeSeriesGroupedTelekomEvent*. This dataset can be injected right into the interactive visualizer program.

TimeSeriesTelekomEvent class.

dataset	Date information in *MM.DD* format [*String*]
dataframe	Time frame mapped from timestamp [*Integer*]
lat	Latitude of the tower [*Double*]
lng	Longitude of the tower [*Double*]

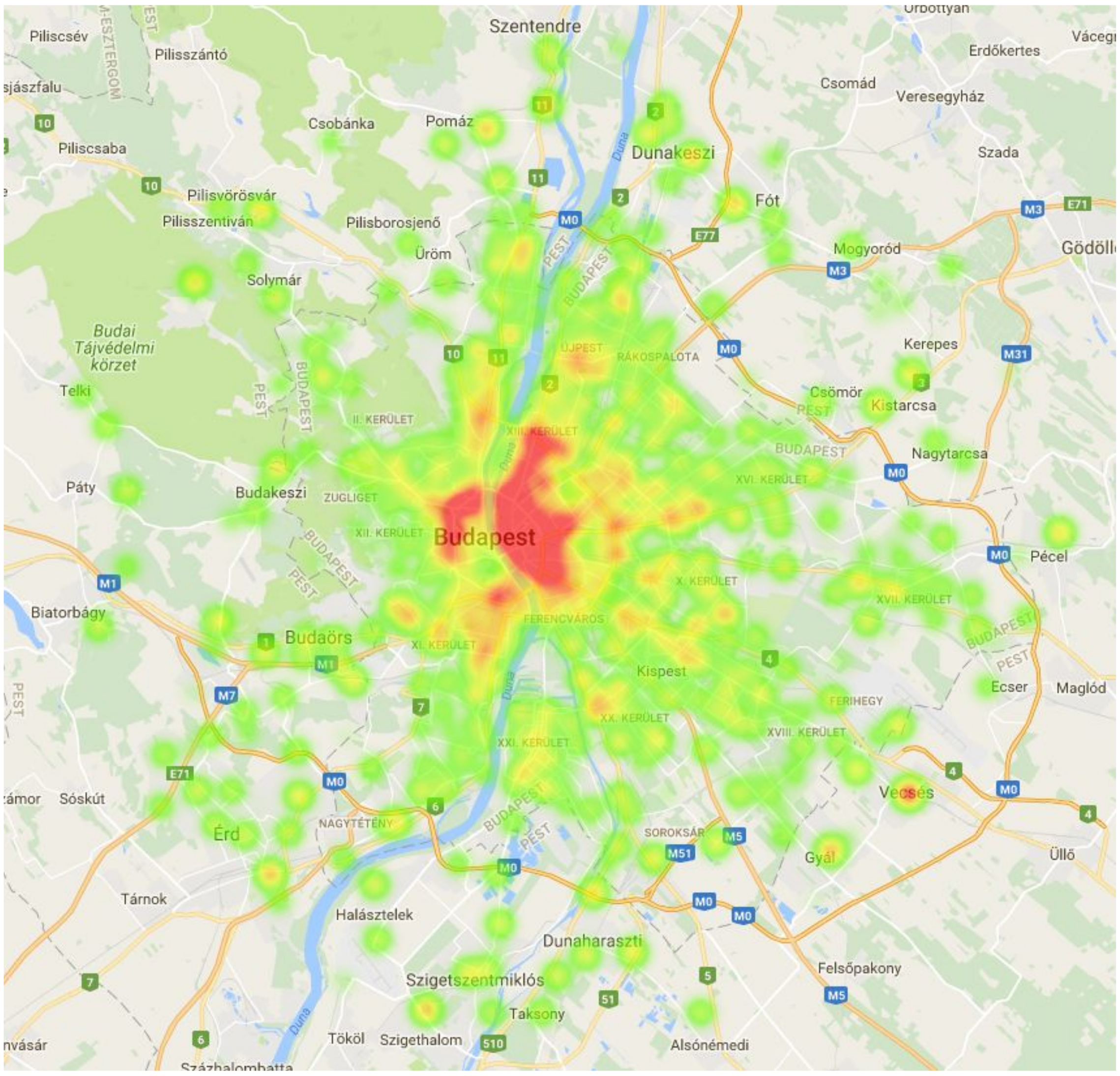
TimeSeriesGroupedTelekomEvent class.

dataset	Date information in *MM.DD* format [*String*]
dataframe	Time frame mapped from timestamp [*Integer*]
lat	Latitude of the tower [*Double*]
lng	Longitude of the tower [*Double*]
crowd	Sum of telco events [*Integer*]

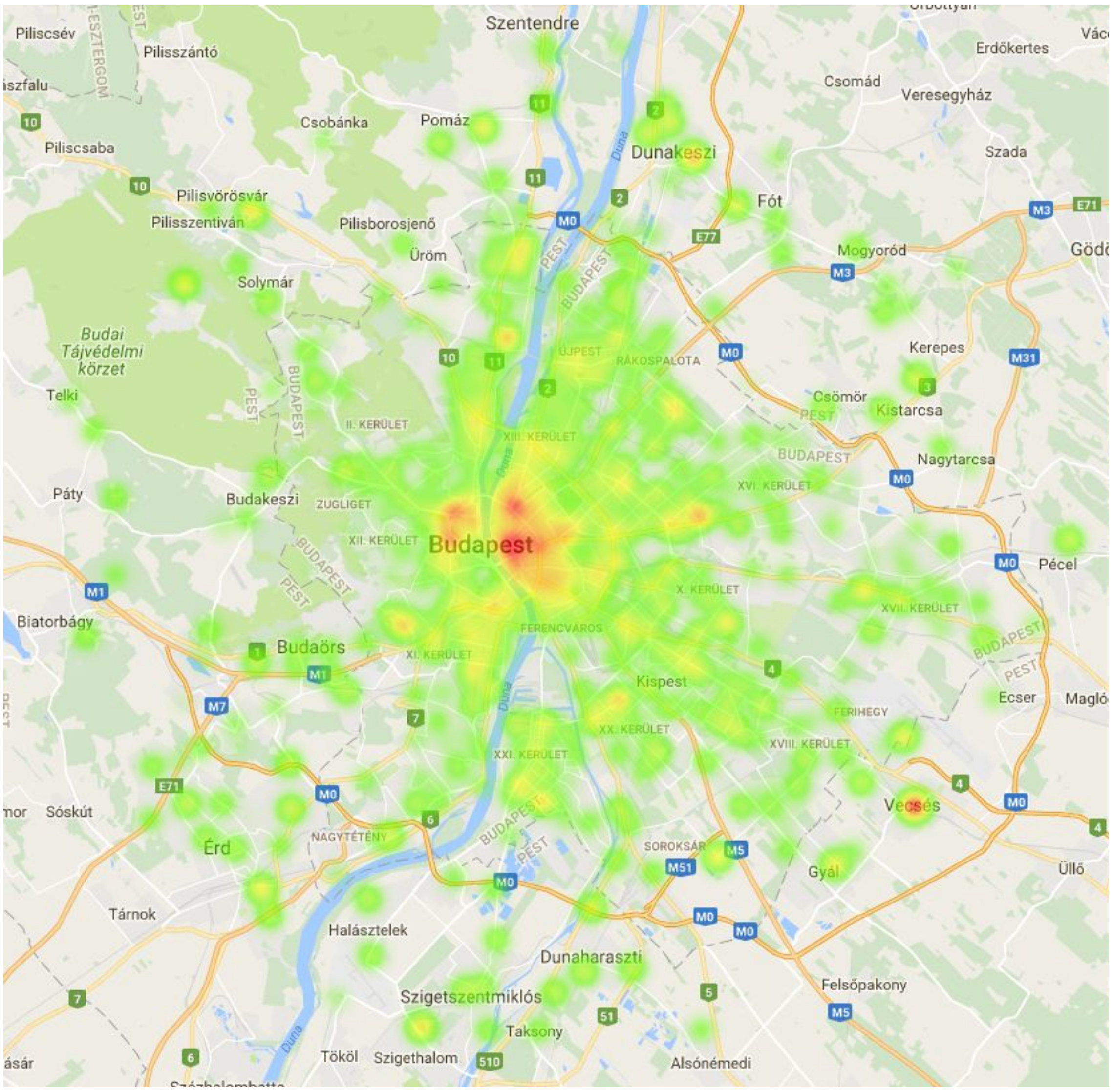
The extent of this dataset will be relatively small with *rows = d * f * c*, where *d* is the number of days (7 - sample dataset, 30 - real dataset), *f* is the number of frames per day (24) and *c* is the number of cells (~2400 - sample dataset and ~3700 - real dataset). In consideration of the small size and the fact that this exact dataset will be reused later in the crowd prediction analysis it was cached into memory to hasten later computing.

7.2 Crowd analysis findings

Let me start with something unsurprising. Compare a Friday’s (23.09.) morning activity with Saturday’s morning activity (24.09.). Friday being a workday and school day morning crowd should be much higher. Let’s use the 7th time window (7 am-8 am) from both days at the outer Budapest area.

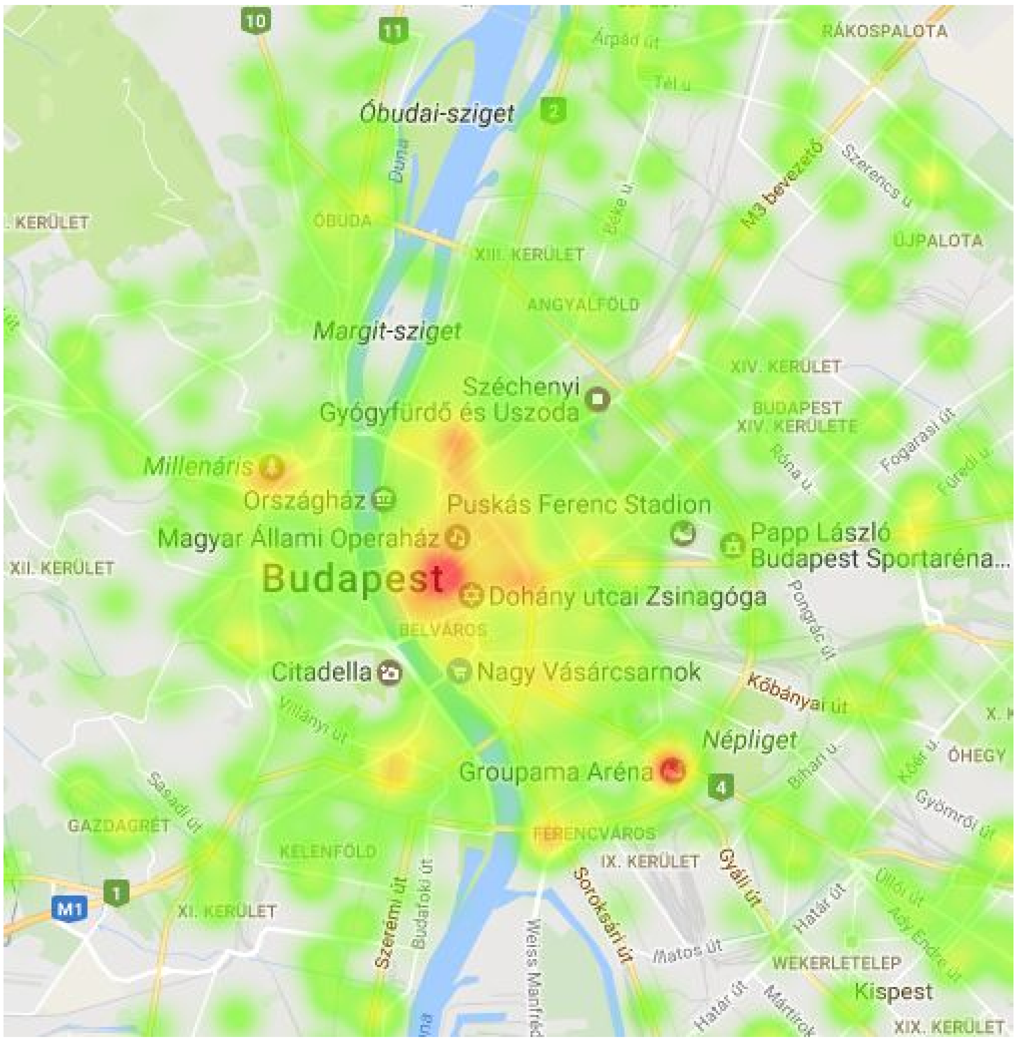


Crowd activity visualized with heatmap on a Friday morning. Red equals “trending” areas, while greens show less crowd.


Crowd activity visualized with heatmap on a Saturday morning.

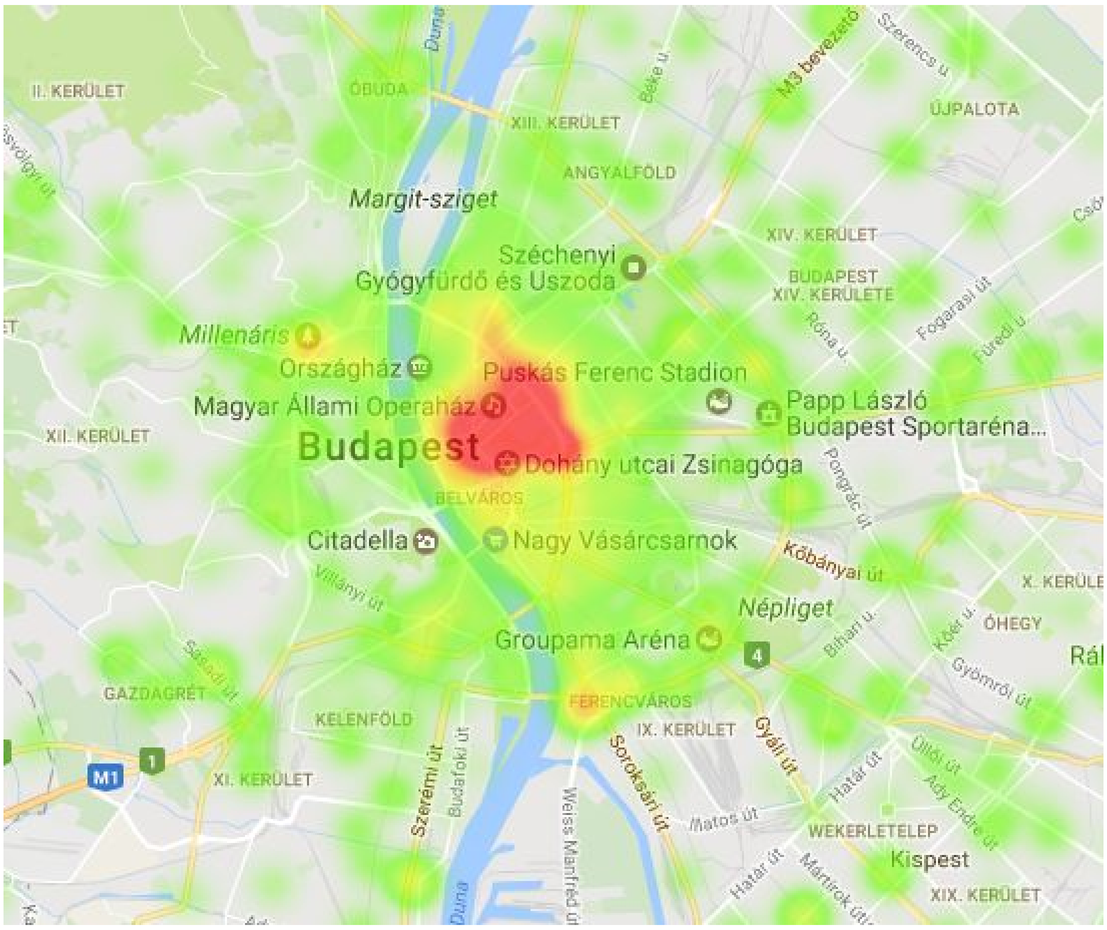
The difference is significant in the center areas of the city, but a lot more marginal in the outskirts of the city. Vecsés (Ferihegy Airport) and other suburban areas had close to identical crowd.

Let’s examine what happens during and after an event - a football match here. FTC-UTE match took place on the 24th of October at 8:30pm, Groupama Arena, which is basically the hungarian *el clasico* and attracts a lot of visitors.



Snapshot of the crowd activity on the match's evening from 8pm to 9pm. Notice the crowd concentration bottom-right from the center. Such crowd hadn't been seen during the two week time period at the same place in any other time.

There are a lot of telco activity going on at that area - not a big surprise. What is more interesting is that which part of the city is this crowd heading to after match? Did they just go home?



Snapshot of the crowd activity on the match’s evening after match between 11pm and midnight.

There you have it. It is the crowd from the match or just people enjoying Budapest’s nightlife? The downtown area seems similarly populated on the other Saturday (17.09.). There is increased activity on the east perimeter of the downtown area (“Kiskörút”), where there are a myriad of sport-related pubs. This might be activity from the people who were enjoying the match in one of these pubs and those people going home or potentially joining them.

8. Crowd prediction

The final and most important chapter of this thesis is the crowd prediction part. As it was discussed before crowd at each cell/tower equals to the number of telecommunication exchanges going on there in that particular time frame. The starting point is having rows with *TimeSeriesGroupedTelekomEvent* class. A predictor needs adequate amount of data and set of important features to be able to make accurate predictions. In the current state only dataset, dataframe, latitude and longitude rows are here as features. This is lacking thus other artificially created variables were introduced. The next section dedicated solely to this *feature engineering* process and further potential improvability. Given features are ready and merged together into a new dataset a good predictor to be found.

Several classifiers and regressors were test:

- Logistic Regression (multinomial)
- Linear Regression
- Generalized Linear Regression (- Poisson Regression)
- Naive Bayes
- Decision Tree Classifier
- Decision Tree Regression

The machine learning algorithms were provided by Spark ML [7] and trained under equal conditions. These are all well-known methods and I refrain myself from going into specific details as there are countless excellent sources available. Just a quick recap: **classifiers** (*Naive Bayes* and *Decision Tree*) work with categorical target data, while **regressors** (*Logistic Regression, Linear Regression*, *GLM* and *Decision Tree Regression*) work with continuous data.

Classifiers have the limitation that they can only predict values that were in the training set. Given each prediction they provide a confidence value that shows how “certain” they are in their prediction for that value. Classifiers work best with a few categorical target values (for instance yes/no outcome). Crowd density is more like a continuous data with ~6-7k possible crowd values from the real dataset. The classifiers worked remarkably well with the heavily sampled sample dataset, because the variance and possibilities were much lower than of the real dataset. Restraining the focus area to very specific parts of the country or a city - however - could yield precise values with the classifiers. Discretizing crowd values into a given set of buckets (e.g. 10 or 100) could potentially have worked, but would have made the comparison with regression models less accurate.

Regressors have no such limitation, but are generally get highly sensitive to outliers and overfitting. They can predict values not have seen before and predict real values instead of categorical. In linear regression features should have minimal correlation with each other. Here latitude and longitude are highly correlated, which causes distortion in the prediction. This will be seen nicely, when working with smaller and smaller area filtering.

For the training and evaluation of the mentioned algorithms the following steps were done:

(1) Same dataset and same features
(2) Divide the dataset randomly into 70% training and 30% test datasets
(3) Train them on the training data with *hyperparameter tuning* [13] (except Naive Bayes) and *4-fold cross-validation* for maximal predictive power. For classifiers the target was **max**(*accuracy)* and for regression **min**(*root mean squared error).*
(4) Predict the test dataset with the trained model and evaluate the prediction based on several metrics:
    - Root mean squared error (*RMSE*) - the lower the better
    - Mean absolute error (*MAE*) - the lower the better
    - R-squared (*R2*) - the higher the better

    - F1 - the higher the better
    - Accuracy - the higher the better
    - Precision - the higher the better

(5) Compare predictive power, training time and most important features per predictor.

8.1 Feature engineering

Feature engineering is essential in building a good predictor. The information we currently possess is a bit lacking: dataset, dataframe, latitude, longitude and crowd. The target variable is the crowd attribute leaving us with four columns as potential features. Without the aid of external data additional attributes have to be derived from the already existing columns. Actually, with a little bit of imagination plenty of ideas arose - unfortunately some did not make to the final cut. Firstly, each tower's previous measured crowd was added to the dataset. Implementing this was a little tricky, because the dataset were partitioned by day so far for parallel computation.

A more creative approach was to include the neighbouring towers to the prediction somehow. For the sake of this a kNN (*k-nearest neighbour*) algorithm was constructed. Crowd information at a given time frame and distance from the current tower are known of these nearest towers. A brute force algorithm ($O(n^2)$) calculated the kNN data for each cell. I could get away with this naive implementation simply, because the dataset of the cells were tiny in the big data world (~3700 rows). That is ~14 million distance computations or to express it in time: a few minutes. If the number of cells would be considerably higher this algorithm would not be applicable.

The following features were added to the existing dataset:

- Min(c_1^j, …, c_k^j)
- Sum(c_1^j, …, c_k^j)
- Avg(c_1^j, …, c_k^j)
- Distance of the closest tower (*in km*)

where c_i (i=1, …, k) is the *i*th closest tower's crowd in the *j*th time frame. The new class became the *KnnPredictionReady* class.

KnnPredictionReady class.

dataset	Date information in *MM.DD* format [*String*]
dataframe	Time frame mapped from timestamp [*Integer*]
lat	Latitude of the tower [*Double*]
lng	Longitude of the tower [*Double*]
*crowd**	Count of telco events [*Integer*] - *target variable*
lastFrameCrowd	Last frame's crowd value [*Integer*]
knnMin	Minimum crowd of the *k* closest tower [*Integer*]
knnSum	Sum crowd of the *k* closest tower [*Integer*]
knnAvg	Average crowd of the *k* closest tower [*Double*]
knnClosest	Haversine distance to the closest tower [*Double*]

The algorithms were run on datasets with *k* = 3. To sum up the goal is to make as accurate predictions for each tower load at given day and time (time frame) as possible by maximizing accuracy and minimizing the error at the same time. In the beginning the neighbouring cell's load was weighted with their distance from the original tower, but later this information was extracted as a new column (*knnClosest*).

The dataset now being bolstered with the shiny new attributes let's see how well the diverse predictors - classifiers and regressors alike - perform.

Which features attribute to the most to the predictive power of each model and how do they perform regarding the given metrics? How accurate are they when predicting nation-wide crowd and the crowd in Budapest? Is there a clear winner or multiple models have to be kept to be leveraged in different scenarios?

8.2 Crowd prediction

The ml algorithms will be compared based on how many good predictions they can output (accuracy, precision, R2) and their error rate (RMSE, MAE).Their values are calculated in different ways and we would like to hit a sweet spot of training time, accuracy and low error measures. The best algorithm is not necessarily better in every listed aspects. Since F1, accuracy and precision makes very little sense in regression analysis - and as it turns out neither in classification - they can be ignored. I have still included it in the first comparison table.

The algorithms will be tested with 3 area filterings:

(1) No filter (= nation-wide dataset)
(2) Outer Budapest (Budapest and suburbs)
(3) Inner Budapest (Budapest downtown area)
(4) Ferihegy Airport Terminal B - *4 towers*

Let’s see how all the work so far come to fruition and run the analysis on the unfiltered dataset. The analysers were ran against the real dataset of 2 weeks. Model metrics, feature importances - where I was able to extract them - and runtimes will also be highlighted in the upcoming tables.

Predictive power - full dataset.

	Logistic Regression	Linear Regression	GLR - Poisson	Naive Bayes	DT Classifier	DT Regressor
RMSE	-	461,23	484	3453,8	599,92	**395,32**
MAE	-	299,51	313,35	3274,36	330,91	**237,69**
R2	-	0,18	0,1	-46	-0,42	**0,4**
F1	-	0	0	~0	0	~0
Accuracy	-	0	0	~0	**0,03**	~0
Precision	-	0	0	0,002	0,02	**0,04**
Runtime	10h+	~ 6 mins	**~ 2.5 mins**	**~ 2,5 mins**	~ 5,3 hours	~ 5 mins

The R2, F1, Accuracy and Precision values are all very low over all predictors. This is not surprising, since there are ~4600 possible crowd values from the data collected over this 2 week. These metrics are based on perfect predictions, which are close to impossible with so many possible target values. For this very reason they will be excluded from further analysis and won't be shown. The Decision Tree Regressor takes a convincing win here despite the fact that it is not the fastest of all. Logistic Regression performed excellently on the sampled set - even on par with the Decision Tree Regressor - but took so long to train I've eventually killed the process after 10 and a half hour. The Decision Tree classifier needed immense amount of time to train compared to the other algorithms as well and still produced the second worst results. The Naive Bayes algorithm was fast, but the results were extremely divergent from the real values. Linear Regression came second and even beaten the GLR Poisson regression - a predictor that performed superbly with the sample set.

Now let's have a look at the feature importances of each predictor - where it could be extracted. The most important features are marked with double stars (**) and useful features with a single star (*).

Feature importances - full dataset.

	Logistic Regression	Linear Regression	GLR - Poisson	Naive Bayes	DT Classifier	DT Regressor
dataset	-	-1,78*	-0,01*	-	0,04	0,08*
dataframe	-	4,61*	0,02*	-	0,47**	0,2**
lat	-	-16,9**	-0,04**	-	0,09	0,06*
lng	-	23,15**	0,07**	-	0,11*	0,01*
LFC	-	0,11	~0	-	0,04	0,03*
knnMin	-	-0,17	~0	-	0,04	0,02*
knnSum	-	0,07	~0	-	0,04	0,35**
knnAvg	-	0,25	~0	**-**	0,05	0,02*
CT	-	-15,64**	0,05**	**-**	0,13*	0,15**

It was surprising to see that how insignificant are the load of nearby cells on the predictive tower and how influential was the nearest tower's distance. The reason for this could be that loads are nicely distributed among nearby cells due to the auto load-balancing nature of the cellular towers. It might have been more useful to see nearby towers' loads in the previous time frame. The last frame's crowd feature definitely added more predictive power, but couldn't be a defining feature. The time-related values made consistent contributions to the prediction in all analyzers. Latitude and longitude values also made a significant addition to the predictive power. Unfortunately I was not able to juice feature importances from the Naive Bayes and the Logistic Regression model.

Latitude and longitude are major decisive attributes in Linear Regression and Generalized Linear Regression, but less important in Decision Trees. Even so it would be interesting to run the algorithms without lat/lng values to get a more general, cell independent predictor. This way a *theoretical* or *planned* tower's crowd/load could be calculated. In this model - however - last frame's crowd feature could not be used. Fortunately, that is not among the most important features, either. Notice how nicely the Decision Tree Regressor could use all features - there are no useless features for it. It could even leverage knn{Min, Sum, Avg} attributes no other algorithms could.

For the reason Logistic Regression and Decision Tree Classifier took so long to train I've excluded them from the next iteration, which is running these algorithms against cellular data only from outer Budapest area.

Predictive power - outer Budapest dataset.

	Logistic Regression	Linear Regression	GLR - Poisson	Naive Bayes	DT Classifier	DT Regressor
RMSE	-	506,12	536,03	2285,5	-	**411,1**
MAE	-	348,6	367,01	1987,05	-	**256,21**
R2	-	0,19	0,09	-16,23	-	**0,47**
Runtime	-	~ 2,5 mins	**~ 1 min**	**~ 1 min**	-	~ 3 mins

I have expected the predictive power to increase with the area-filtering, but the complete opposite thing happened - every predictor lost predictive power, except the Naive Bayes. The reason behind this could be that crowd is more predictable outside the capital (less deviation) making the unfiltered dataset prediction metrics values appear better. I believe the phenomenon behind Naive Bayes being better is simply a coincidence, because it's huge inaccuracy fluctuates a lot based on varying data - a phenomenon that was experienced beforehand during the sample set's prediction.

Feature importances remained roughly the same as seen before with minor motions. The upcoming prediction is based of Budapest's downtown area. I assumed that this dataset is small enough to be handled by Logistic Regression and Decision Tree Classifier and gave them a try.

Predictive power - Budapest downtown area dataset.

	Logistic Regression	Linear Regression	GLR - Poisson	Naive Bayes	DT Classifier	DT Regressor
RMSE	639,51	522,78	535,08	1776,9	504,73	**249,6**
MAE	367,27	364,87	374,82	1539,23	260,4	**144,5**
R2	-0,49	0,15	0,11	-10,52	0,07	**0,81**
Runtime	~3,2 hours	~1 min	<1 min	<1 min	~20 mins	~ 1 min

Now with Logistic Regression and Decision Tree Classifier joint we may see how strong Decision Trees are overall to tackle this problem. Apart from runtime Decision Tree Classifier took 2nd place after its' regressor counterpart. Logistic Regression took really long to train and produced unconvincing results triumphing only Naive Bayes. The GLR and the Linear Regression models are head-to-head with the edge going to the latter. Shrinking the area and the dataset also makes them less and less accurate. They could be viable with much more data and in terms of runtime GLR scales extremely well - better than the Decision Tree Regressor.

Feature importances - Budapest downtown area dataset.

	Logistic Regression	Linear Regression	GLR - Poisson	Naive Bayes	DT Classifier	DT Regressor
dataset	-	-4,07*	-0,01*	-	0,14**	0,09*
dataframe	-	7,01*	0,03*	-	0,17**	0,22**
lat	-	3139,04**	0,05**	-	0,11**	0,16**
lng	-	1665,22**	0,05**	-	0,14**	0,16**
LFC	-	0,2	~0	-	0,12**	0,04*
knnMin	-	0,35	~0	-	0,09*	0,04*
knnSum	-	0,03	~0	-	0,08*	0,01*
knnAvg	-	0,1	~0	**-**	0,05*	0,09*
CT	-	229,2**	0,23*	**-**	0,11**	0,2*

Logistic Regression returns a coefficient matrix that is different for every label thus the feature importances are not obvious to get. The Linear Regression and the GLR model is prone to be skewed by some highly important features or correlations and could not take advantage of the others. Correcting this with outlier filtering and merging lat/lng to unique numerical values might make these models much more viable, though.

Narrow down the dataset to the Airport area only now. This is a 1 km radius circle with 4 telco towers in it. If anywhere, this is the place for the classifiers to show strength.

Predictive power - Airport dataset.

	Logistic Regression	Linear Regression	GLR - Poisson	Naive Bayes	DT Classifier	DT Regressor
RMSE	216,81	719,26	647,77	216,49	155	**105,91**
MAE	142,81	496,03	451,35	144,6	87,73	**33,33**
R2	-0,77	0,59	0,67	-0,76	0,1	**1**
Runtime	<1 min	<1 min	<1min	<1 min	~ 1 min	<1 min

The runtime no longer plays a key role here, since all predictors summarized ran for less than ~3 minutes. Finally Logistic Regression and Naive Bayes bypassed Linear Regression and GLR - both which are basically useless with such limited dataset. The Decision Trees are still going strong here, but Logistic Regression and Naive Bayes have closed a huge gap. After all, Decision Tree Regressor is the way to go independently of targeted area’s size. Lastly, let’s look how the feature importances transformed over our extensive shrinking process.

Feature importances - Airport dataset.

	Logistic Regression	Linear Regression	GLR - Poisson	Naive Bayes	DT Classifier	DT Regressor
dataset	-	2,65	0,01	-	0,36**	0,03*
dataframe	-	16,68*	0,02	-	0,18**	0,06*
lat	-	-1650715,2**	-0,07*	-	0,06*	0,02*
lng	-	715688,31**	-0,11*	-		
LFC	-	-0,18	6**	-	0,11**	0,25**
knnMin	-	8,27*	0,02	-	0,15**	0,08*
knnSum	-	1,93	0,03	-	0,1**	0,02*
knnAvg	-	-5,78*	-0,1*	-	0,04*	~ 0
CT	-	-1168,9**	-3,55**	-		0,55**

There are some extreme values at the Linear Regression. One of the reason behind this must be the high correlation between latitude and longitude values. The GLR Poisson is much more balanced, but still a weak predictor compared to the others. The Decision Tree models excluded the longitude values from the feature set, because it provides no additional contribution to the prediction. What made the closest tower's distance to have so low value in the prediction that the Decision Tree classifier have vanished it is beyond me. In other models it is such a valuable asset.

To draw the concise conclusions use Decision Tree Regressor most of the time (and try to upgrade to Random Forest Regressor), GLR is the way to go with enormous amount of data and preference of speed over predictive accuracy and classifiers or Logistic Regression are a viable alternative with immensely focused area of interest. Speaking of features the knn* features do not bring much to the table, but far from being useless. The closest tower's distance did surprisingly good and encourages more distance-related data to be drawn into the prediction.

9. Summary and future work

Writing this thesis gave me invaluable knowledge about telecommunication data & infrastructure, crowd movement, big data analysis and machine learning algorithms in a large-scale environment. The tasks were fun, challenging and often required creativity. The groundwork is laid for a lot more to come with this data as there is a huge potential in it. The outcome framework - backend and frontend - has the ability to analyze and visualize telecommunication flow and various mass movement metrics.

Apache Spark proved to be an excellent framework for big data jobs and it has gained a lot of popularity over the past few years, so I am satisfied with my choice. The only letdown was that I could not run my framework on a clustered environment. For the sake of data security I could not just fire up an AWS cluster to run and benchmark it, all the work had to be done on one isolated machine. Still, the application scaled up really well to 8 cores/16 threads - monitoring the CPU showed constant 95%+ usage over all cores. Scala and its functional style and ecosystem was also something new and refreshing and was a pleasant overall experience.

I have gained a deeper understanding of the used machine learning algorithms, their strengths and limitations, application areas and the metrics how they can be evaluated.

I have a bittersweet feeling about this work. From one point of view a lot of work was put into it, but on the other hand a lot more could have been done and the interesting part would only begin now.

Counteractions against CDR data inaccuracy (biased crowd data), analysation of more complex paths (e.g. freeways) rather than circular areas only. File formats designed specifically for big data (apache avro, apache parquet), pulling in external data (e.g. weather conditions), work with a bigger time window (1 month+ data). Extract more simple features for the predictors (e.g. day of the week) and put more advanced ml algorithms to test like Random Forest (regressor), Gradient-boosted Tree (regressor) and ensemble models. Outlier detection was also out of scope. I think these models can achieve lot better results when fully optimized. This thesis was just a kick off to see which direction to head for, if any.

I genuinely think that a fine-tuned regression model (Decision Tree Regressor, Random Forest Regressor) would be a first class method to predict future tower loads and crowd at the same time with fast execution.

The source code is available in a private github repository here: [14]. If you need access rights feel free to contact me.

References

[1] Richard A. Becker, Ji Meng Loh, Ramón Cáceres, Margaret Martonosi, Alexander Varshavsky, Karrie J. Hanson, James Rowland, Chris Volinsky, Sibren Isaacman, Simon Urbanek - Human Mobility Characterization from Cellular Network Data, 2013

[2] Ilyas Alper Karatepe, Engin Zeydan - Anomaly Detection In Cellular Network Data Using Big Data Analytics, 2014

[3] Telekom coverage, http://www.telekom.hu/lakossagi/szolgaltatasok/mobil/lefedettseg

[4] Salvatore Catanese, Emilio Ferrara, Giacomo Fiumara - Forensic Analysis of Phone Call Networks, 2013

[5] Apache Spark, http://spark.apache.org/

[6] Apache Spark SQL, http://spark.apache.org/sql/

[7] Apache Spark ML, http://spark.apache.org/docs/latest/ml-guide.html

[8] In-depth guide to Apache Spark Catalyst Optimizer, https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html

[9] Google Maps Javascript library and API, https://developers.google.com/maps/documentation/javascript/

[10] Apache Spark Programming Guide, http://spark.apache.org/docs/latest/programming-guide.html

[11] Haversine formula in Scala,

https://rosettacode.org/wiki/Haversine_formula

[12] Haversine formula, http://www.movable-type.co.uk/scripts/latlong.html

[13] Machine learning hyperparameter-tuning,

https://spark.apache.org/docs/latest/ml-tuning.html

[14] https://github.com/brajer/TelekomAnalyzer