UiT THE ARCTIC UNIVERSITY OF NORWAY

The Faculty of Science and Technology
Department of Computer Science

Scalable exploration of population-scale drug consumption data

—

Tengel Ekrem Skar

INF-3981: Master's Thesis In Computer Science
June 2019

Abstract

The potential for knowledge discovery is currently underutilized on pharmacoepidemiologic data sets. A big dataset enables finding and assessing rare drug consumption patterns that are associated with adverse drug reactions causing hospitalization, or death.

To enable such exploration of big pharmacoepidemiology data, four key issues need so be addressed. First, to ingest, transform, preprocess and analyze population scale data, we require large computation power and storage capabilities, and therefore a distributed computing framework. Second, to expose patterns between drug consumption and end-points such as hospitalization, we need to develop feature extraction and preprocessing algorithms which represents the drug consumption and hospitalization in a numerical format. Third, to detect these patterns, we require models from libraries for statistics and machine learning. To interpret performance metrics, we also require visualization libraries. Fourth, to enable rapid development of data exploration methods, we require an interactive system that makes the frameworks, libraries and methods for explorative analyses available in a single, cohesive environment.

We make three contributions.

First, we present the design and implementation of a system with a live coding environment, which enables use of Apache Spark, our choice of big data framework. It provides Scikit-learn and Tensorflow with Keras for machine learning, and matplotlib and Plotly for visualization. All libraries and frameworks are made available by the interactive environment, which enables rapid development, and Spark enables workloads to scale.

Second, to enable machine learning methods, we provide algorithms for feature extraction of drug consumption. We observe drug consumption in hospitalized and unhospitalized patient groups, and label them according to their group. This results in a data set that we use in supervised learning.

Third, we assess the performance in prediction of hospitalization on the data set. We also estimate over-represented drugs in hospitalized patients.

The results are available in an executable notebook format, and the implementations are modifiable so that researchers can re-purpose the preprocessing algorithms and analyses for their needs.

To predict hospitalization, a logistic regression achieved an Area Under the receiver operating characteristic Curve (AUC) of 0.758, and a neural network achieved an AUC of 0.771.

We bootstrapped logistic regressions to obtain a list of 200 (of 900) drugs that the regression obtains stable estimates for. The omitted 700 drugs had high variance, which indicates that they are under-represented in our data altogether.

The predictive performances were not very good. From the bootstrap analysis we identified which drugs occur frequently enough in our data, and which don't. We believe that improved data cleaning can improve both models prediction performance. We believe more data will enable more accurate log-odds estimates for the remaining 700 drugs. We learned that good prediction of hospitalization from drug consumption isn't possible with our current preprocessing, but we also learned which drugs that are most and least likely usable for prediction.

Acknowledgements

First, I want to express my deep gratitude to my supervisors, Lars Ailo Bongo and Kristian Svendsen, for their advice, feedback and constructive criticism throughout this work. I would also like to thank my co-supervisor, Einar Holsbø, for his great statistical knowledge, constructive criticism and discussion, which inspired and shaped the methodology in this work. None of this would be possible without all three of you.

I would like to thank my good friends and classmates, Maren, Andreas, Nikolai, Sverre and Vebjørn for the enlightening technical discussions, exchanges of knowledge, and great banter.

Thanks to my long-time friend and computer scientist Håvard Ola, for proof-reading and suggesting minor changes to figures, citations and structure.

Maren, thank you for your love and support. You made the most challenging periods of writing bearable.

Contents

List of Figures

List of Tables

1 Introduction

There has been large advances in distributed computing and machine learning methods and frameworks for big data analyses. These have mostly been developed and used by the tech industry. The frameworks enable exploration of datasets which were historically too large, complex and computationally intensive to analyze with traditional tools and methods. However, developing analyses with these frameworks requires a combination of expertise in distributed computing, machine learning and statistics that many organizations do not have. Therefore the potential for knowledge discovery is currently underutilized on big datasets in many fields.

One field with underutilized datasets is pharmacoepidemiology, which is the study of consumption and effects of drugs on populations. An important aim of pharmacoepidemiologic research is to assess the risks of adverse effects from combinations in thousands of different drugs. Investigating these risks on a population often requires finding drug consumption patterns in groups, by linking drug consumption with known outcomes such as adverse drug reactions, hospitalization and death . The main challenge is that detection of drug combinations which occur rarely in a population require enormous amounts of data, and thus new approaches to explore and analyze it must be developed.

To enable exploration and analyses of big pharmacoepidemiology data, four key issues need so be addressed. First, we need frameworks which leverage large amounts of computing power to ingest, transform, preprocess and analyze the data. This requires a system for distributed computing. Second, to explore data, we require numerical libraries for statistics and machine learning, and libraries visualization libraries. Third, to find new patterns in the data, we need to develop methods for feature extraction, prediction and to estimate over-representation of drug consumption in patient groups. This includes preprocessing schemes to transform prescription data to labeled training data, which in turn is used to train supervised machine learning models. Once a model is trained, we also need methods to interpret the model and its performance. Fourth, exploration requires a system which enables quick development of these methods. Therefore we require an interactive system that makes the frameworks, libraries and methods available in a single environment.

We make three contributions. We present the design and implementation of a system with a live coding environment, which enables use of Apache Spark, our choice of big data framework. It provides Scikit-learn and Tensorflow with Keras for machine learning, and matplotlib and Plotly for visualization. We provide algorithms for feature extraction, which we use to link drug consumption with hospitalization. We assesses the performance in prediction of hospitalization on these linked data sets. We also estimate over-represented drugs in hospitalized patients. The results are available in an executable notebook format, and the implementations are modifiable so that researchers can re-purpose the data and analyses for their work.

1.1 Problems

1.1.1 Problems with using existing data analysis systems

Tools such as Excel, Stata, Python and R are commonly used to explore and analyze datasets up to a few gigabytes in size. Their workflows are great for fast data exploration because they have a large set of packages for data analysis available, and most offer some interactive coding environment. They do however face scalability issues when the data outgrows the memory or processing power that a single machine provides.

A big data framework such as Apache Spark [1] can seamlessly scale analyses by adding more machines. Spark was originally designed as an alternative to MapReduce [2, 3] based frameworks. It reaps large performance gains by favoring in-memory processing over MapReduce's step-wise operations with intermediate results written to disk.

However, Spark does not provide the functionality required for machine learning. Its machine learning library, MlLib is not very mature, and the evaluation metrics [4] available for its machine learning models are limited. Second, Spark has no built-in tools for visualization. Therefore Spark is often used in combination with a machine learning framework.

1.1.2 Challenges for data exploration

In previous work, we implemented a combinatorial analysis where we aggregated drug consumption as time-based data on a per-patient basis. The result showed which drugs are used together in combinations of two. However, these became difficult to interpret past combinations of two, because the resulting output scaled combinatorially with the number of unique drugs. With combinations of two and a 856 unique drugs it resulted in a 856X856 matrix as output.

Instead, a supervised machine learning model can be trained to learn correlation between drug consumption and outcomes such as hospitalization, adverse drug reactions or deaths. To apply supervised machine learning four issues must be addressed.

First, our pharmacoepidemiologic data sets contain events such as drug consumption, hospitalization, death and treatment duration for our population. It therefore has a large number of correlations between the different events. We need to find out which of these patterns we can use as predictors and outcomes in the data.

Second, the predictors and outcomes must be preprocessed to select features representing the patterns in the data. The feature space should exclude post-outcomes to reduce overfitting.

Third, we need a machine learning model which learns patterns in the preprocessed data. The type of model determines how complex relationships it can learn between the features in the training data and the outcome, and increasingly complex models have higher chance of overfitting, and worse interpretability than simpler models. A

Finally, to extract new insights about pharmacoepidemological data, we must choose interpretable methods to assess our models. These results should be visualized so pharmacoepidemological researchers can assess their usefulness.

1.2 System and methods for data exploration

To enable high development speed in pharmacoepidemiological data exploration, we compose a system using existing tools, that provides the following features.

The system enables interactive code execution with a notebook environment. The environment has a framework for distributed computing available, and also provides libraries for preprocessing, machine learning and visualization. The notebook environment enables the developer to compose all steps, from preprocessing to analysis inside with a set code blocks. Comments can be written in markdown, and results are displayed below each cell.

Using a notebook enables data exploration because it tightly integrates development, execution and evaluation of analyses in the same runtime, making exploration fast. The environment enables reproduction of results, because the full analysis - from raw data to result - can be fully contained in a notebook.

Third, the system provides methods for loading, storing, transforming and querying large data so that analyses can be performed on it. This is provided by the distributed computing framework. Data is manipulated with SQL-like querying on tabular data, which is an accessible and powerful feature that makes data transformations of pharmacoepidemiological datasets much quicker to develop than with low-level alternatives such as MapReduce.

Fourth, the framework provides tools for numerical computing, visualization and machine learning. This enables the user to apply pattern recognition algorithms on the data, and to interpret results of analyses.

As a starting point for further pharmacoepidemiological research, we implement a set of algorithms to preprocess and analyze prescription data:

1. We convert the raw tabular data to tables in the Parquet [5] format such that simple exploration scripts can be implemented using Spark SQL.

2. We provide feature extraction of drug consumption from prescription data sets.

3. We allow adjusting the dimensionality of the feature space by adjusting the number of Anatomical Therapeutic Chemical Classification System (ATC) levels.

4. We compare the performance of a logistic regression and neural network

for prediction of hospitalization based on our drug consumption features.

5. We use the bootstrap to obtain a list of drugs that are over- and under-represented in hospitalized patients.

6. We enable reproducing or improving our analyses by providing the descriptions, code, visualizations and documentation for our analyses in notebook format.

1.3 Summary of results

Our big data analysis system provides the functionality required to efficiently explore and develop new algorithms on pharmacoepidemological data of Norwegian elders, with 60 million prescriptions and 2 million hospitalizations. Also, our analyses have demonstrated to work with the available data.

On a single machine, we can preprocess the data of all patients from a single gender in approximately 7 seconds. Training of a logistic regression model on this data takes 10 seconds. Training of a neural network with scikit-learn on CPU takes 400 seconds with one hidden layer with 50 neurons.

As can be seen in Figure 1.1 the neural net and logistic regression models achieve almost identical receiver operating characteristic (ROC) curves and AUC.

In our bootstrapping scheme, we draw samples randomly with replacement from our population. This sampling with preprocessing takes on average 7 seconds. Using Scikit-learn, each logistic regression model takes on average 20 seconds to complete training on a bootstrapped sample. Training of 2000 bootstrapped logistic regressions - sampling, preprocessing and training for each iteration - took 15 hours and 12 minutes to complete on a single machine. The scheme is embarrasingly parallel, so the task should scale linearly if distributed. However, some implementation details must be changed to enable the whole analysis to run on a cluster.

Through bootstrapping of logistic regression models we obtain estimates for the mean and variance of every parameter. These parameters each correspond to the log odds ratio that when a drug is used, a hospitalization occurs. We sort these the parameters by their signal to noise ratio, and find a set of drugs that are either over or under-represented in the hospitalized elders (1.21.2).

From the bootstrap analysis, we learned that many drugs don't occur in enough

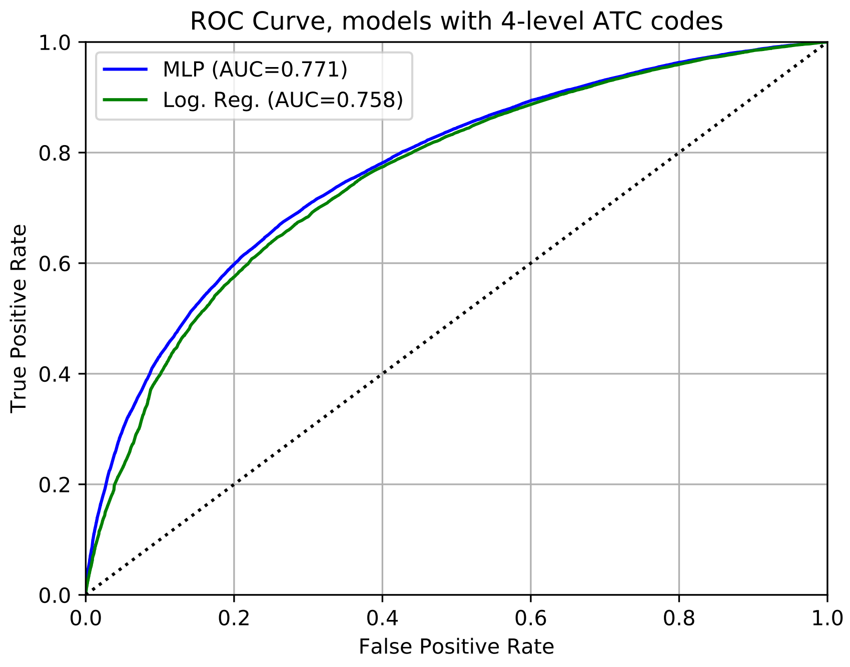

Figure 1.1: Receiver operating characteristic curve of logistic regression and neural network on 4-level ATC codes

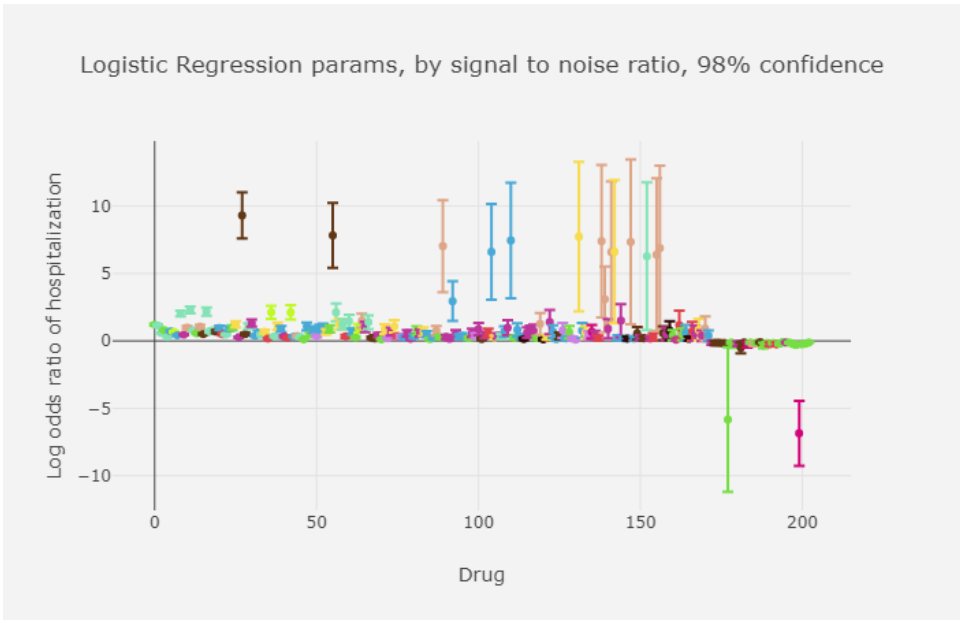

Figure 1.2: Filtered bootstrapped logistic regression parameters, 98% confidence intervals. We omit drugs where the 98% confidence intervals overlap with the log odds equilibrium (0).

observations to obtain consistent log odds estimates. To learn more about these drugs we believe more data is required. We believe that the system and methods we have developed will work on larger data sources.

1.4 Thesis structure

The rest of this thesis is structured as follows. Chapter 2 describes the the design and implementation of our data exploration system. Chapter 3 describes our data sets. Chapter 4 describes the preprocessing and machine learning algorithms we have developed, and our results. In Chapter 5 we provide a performance analysis of our bootstrap algorithm. In Chapter 6 we summarize the contributions of this work. We discuss the limitations of our system, our results, conclude and suggest future work.

2

A system for exploration of pharmacoepidemiological data

2.1 Architecture

The system architecture (Figure 2.1) is centered around a live code engine that the user interacts with by writing code in an interactive programming environment and by interacting with resulting visualizations. The live code engine connects all frameworks, tools and libraries. It enables the user to leverage multiple frameworks and libraries, and therefore implement and execute an entire analysis in a single environment. This reduces development time, and also reduces execution time since code changes do not require re-executing the entire analysis. The code engine integrates the following three features into one environment.

First, the environment connects to a big data framework upon intialization. The big data framework connects the user to the data sources. The framework enables the user to select, transform and aggregate big data using SQL-like primitives on tabular data. SQL greatly simplifies data selection compared to map-reduce based alternatives, making exploration fast. Data can also be pulled from the big data framework into the live coding engine on-demand.

Second, the environment provides libraries and framework for efficient numerical computing, statistics and machine learning. This enables quick development of algorithms for pattern detection on pharmacoepidemiological data. To enable data modeling with machine learning libraries that run inside the code engine, data is be pulled from the big data framework. For machine learning frameworks that run on a cluster (for example TensorFlow clusters), data can be pulled directly into the machine learning framework, from the big data framework.

Third, the environment provides libraries for visualization. This enables interpretation of the results that are obtained from analyses.

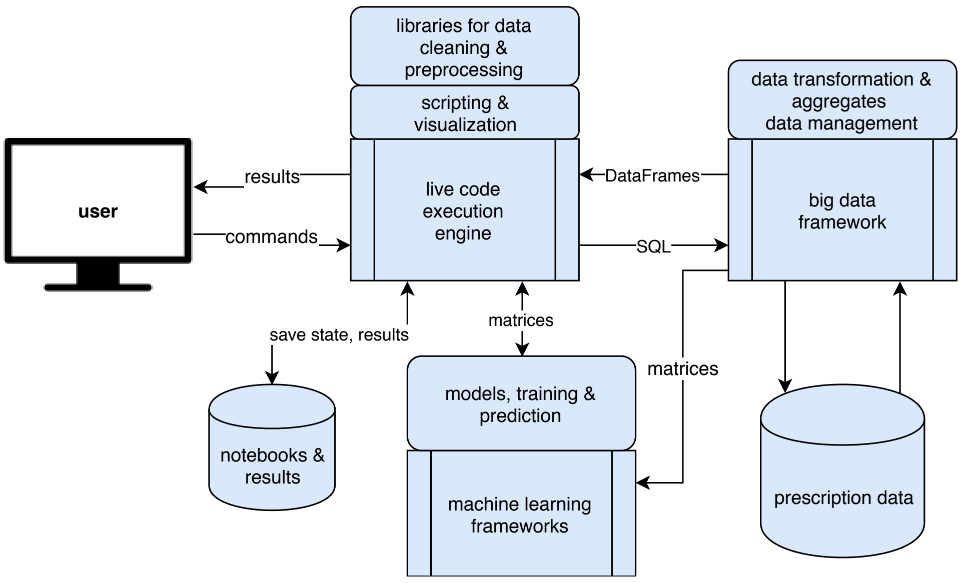
libraries for data cleaning & preprocessing
scripting & visualization
data transformation & aggregates data management
user
results
commands
live code execution engine
DataFrames
SQL
big data framework
save state, results
matrices
matrices
notebooks & results
models, training & prediction
machine learning frameworks
prescription data

Figure 2.1: General data exploration system architecture

2.2 Implementation

We use Jupyter notebook [6] to provide interactive coding. As can be seen in Figure 2.2 it is the core of our system. Our choice of big data framework is Apache Spark [1, 7] with Spark SQL's DataFrame API [8]. This provides a programmatic SQL-like interface to the data sets.

To enable easy access to Spark with as little configuration as possible, we choose Spylon-kernel [9] as the main kernel in the Jupyter environment. On start, spylon-kernel initializes a Spark Session. The Spark Session connects to an existing Spark cluster if Spark is configured, otherwise it creates a virtual Spark cluster which runs on the same machine as the Jupyter environment.

Spylon-kernel has two language interpreters available; a Python interpreter, and a Scala interpreter. Both are connected to the same Spark Session. The Scala interpreter enables performant computations in Spark. The Python interpreter provides a wide array of libraries for analyses, machine learning and visualization.

Our chosen libraries for machine learning are scikit-learn and Tensorflow with Keras. Sci-kit provides a large number of algorithms for preprocessing, and supervised and unsupervised learning. Tensorflow provides great support for development of deep learning models, including support for GPU acceleration and distributed training. Tensorflow also recently got native support for Keras, which is a user-friendly API for machine learning. These libraries enable definition, training and assessment of various machine learning algorithms. For visualization we use matplotlib and Plotly.

The user connects to the Jupyter web server through a web browser. The web server is hosted on a workstation. When connected, the user sees a directory which contains all notebooks, datasets, utility code and plots. Some notebooks contain the data ingestion programs, and the user has to run these the first time the system is set up. When creating a new noteboook, the user has a choice between two kernels, the Spylon-kernel, or the Python 3 kernel.

In Spylon, the user can load all data sets by importing and running some utility functions we have implemented. This registers all Parquets(tables) with pharmacoepidemiologic data so they're queryable through the Spark Session. This makes it possible to query them in the notebook with with Spark SQL. From here, the user may transform, aggregate or pull data from Spark into the Jupyter environment. To do this, the user can use the *DF.toPandas()* function, which transforms the Spark DataFrame into a Pandas DataFrame in Python. Pandas dataframes provides support for tabular data, and are conceptually similar to Spark DataFrames. They also provide great support for interfacing with libraries such as Numpy, scikit-learn and visualization libraries. The user may convert columns or rows in the Pandas DataFrame to numpy arrays easily. This enables the user to easily apply preprocessing and visualization techniques.

With the Python 3 kernel, unlike Spylon-kernel, the user must set up the Spark environment manually. We therefore suggest using Spylon-kernel for any work that involves Spark.

For visualization of results, we suggest storing the results on disk and using the Python 3 kernel instead of spylon-kernel. This is because Spylon requires some workarounds to display graphs, and it has bad support for Plotly.

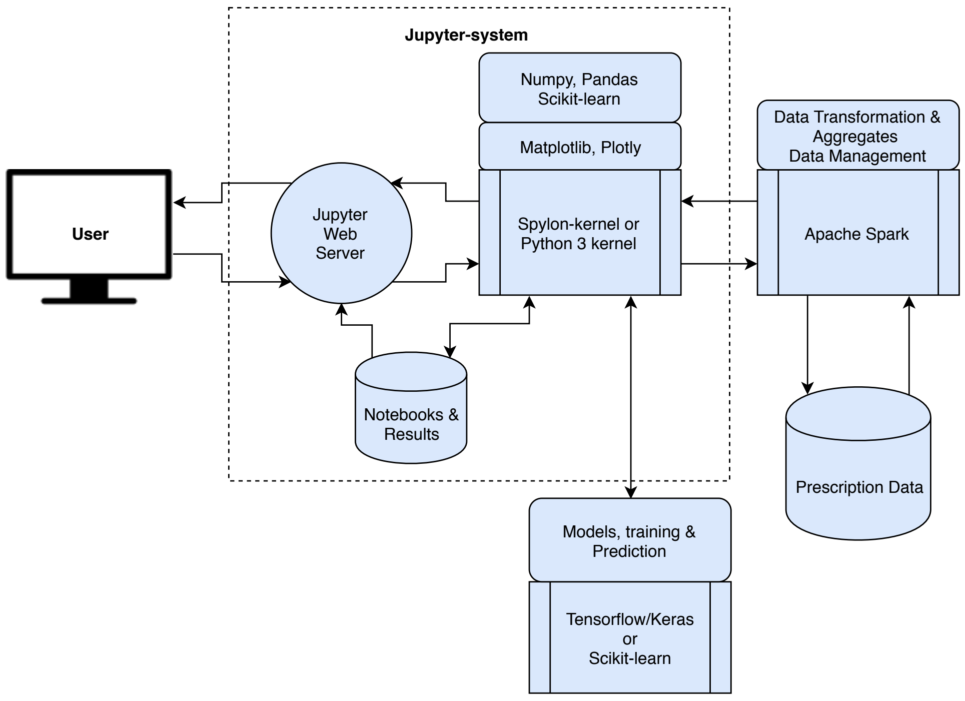

Figure 2.2: Our system design

2.3 Discussion

2.3.1 Apache Spark

We initially decided to use Spark because we struggled working with giga-scale datasets with only Python and Pandas, programs would crash due to memory issues. Using Apache Spark to ingest the data first solved these issues automatically.

Spark versus Hadoop MapReduce

We believe Apache Spark is a better alternative for data exploration than Hadoop MapReduce for a number of reasons.

Our data sets are structured, and Apache Spark has extensive APIs for working with structured and semi-structured data. Apache Spark provides Spark SQL, which enables relational processing on the data. The user writes highly expressive queries, and lets Spark figure out how to actually compute the result. On the other hand, MapReduce implements a low-level programming model where the user defines workloads with a chain of only two different operations. Development of analyses with map-reduce is slower than Spark SQL because the user must decompose his query to a set of steps that fits the paradigm.

Apache Spark achieves better performance in most tasks due to its design. Apache Spark is built on Resilient Distributed Datasets, a distributed data structure which favors RAM over disk use. As long as a data set can fit in the Spark cluster's working memory, it never suffer the bottlenecks experienced by disk I/O. On the other hand, Hadoop MapReduce is optimized for data that is too large to fit in RAM, and uses disk for each intermediate computation.

Today, Hadoop MapReduce has one main advantage in some cases; cost. It can be cheaper to run batch jobs in Hadoop MapReduce because it requires fewer worker nodes to handle large data. The largest tradeoff is generally computation time, Map Reduce jobs take longer to complete.

2.3.2 Alternative notebook environments

There are a number of available notebook environments available. Apache Zeppelin is one such environment. It provides many of the same features as our Jupyter environment with spylon-kernel, and is a viable option to our solution.

However, we ended up using Jupyter because when we designed our system, the general notion in the data science community was that Jupyter was more mature than Apache Zeppelin. However, the Zeppelin community has been steadily growing since then, and the system is built with Spark-support as a main goal. Therefore we believe that it is as useful as our current environment. If we adopted this system, we would only need to substitute the Jupyter environment with Apache Zeppelin. The available libraries in Python would remain the same, and the overall structure of our system would be identical.

2.3.3 Alternative Jupyter kernels

The Jupyter ecosystem has a large number of kernels available. We chose spylon-kernel because it fit our problem space by providing easy methods to interact with Spark. However, the spylon-kernel project is no longer actively maintained, and it hasn't seen updates for months. While we used spylon throughout our work, we would consider migrating to an alternative kernel in the future.

Apache Toree is a kernel which provides a larger set of features than Spylon-kernel. In addition to Python and Scala, it has support for R programming and SQL queries on DataFrames. We investigated possibilities of migrating to Apache Toree. However, we encountered some issues with getting it up and running. Since we hadn't had any issues with spylon, we decided that it was better to spend time on analyses. If Apache Toree was adopted, it would only replace the spylon-kernel, and the rest of the system design would be identical.

2.3.4 Conclusion

Our system provides tools which enable fast data exploration. There are multiple alternatives to every component, but we believe that our chosen solution is sufficient. To target potential issues that may arise with spylon-kernel, we suggest a re-evaluation of the notebook system some time in the future. All the discussed alternatives provide notebook environments that are very similar to our solution. We therefore expect that our analyses should be portable to other systems with minimal implementation changes.

3 Pharmacoepidemiological Data

Norway has a range of high quality health care data sets. The Norwegian prescription database is one such data set. The database contains all drugs dispensed at pharmacies in Norway. Big data analyses have never been used on these data.

Detection of patterns such as drug interactions require development of analyses that process huge amounts of data. Such analyses involve linking drug consumption with data on hospitalization, death or adverse drug reactions. An adverse drug reaction is an injury that is caused by taking medication, and can occur from consumption of a single dose of a drug, use over time, or by interactions from consuming a combination of multiple drugs. This is called comedication. Detection of ADRs caused by comedication is a complex issue. It is not guaranteed that the health care system will detect the cause of the adverse effect when it occurs. If two drugs which cause adverse reactions when comedicated are rarely prescribed, this requires huge amounts of data to achieve a satisfying confidence.

3.1 Introduction

Our research utilizes drug-consumption related data from multiple Norwegian sources. We have three data sets available: two data sets from the Norwegian prescription database (NorPD), and one data set from the Norwegian Patient Registry, containing hospitalizations. The two data sets about elders are linkable by the patient's ID. This enables many types of analyses. For example analyses of correlation between drug consumption and outcomes such as hospitalization and death, but also studies of doctors and how they medicate their patients. The available data sets are listed in Table 3.1

Name	Size	Information
NorPD All	375 Million	Prescriptions
NorPD Elders	60 Million	Prescriptions+more
NorPD Elders Treatment Duration	60 Million	ATC, start+end
NPR, Elders	1.9M hospitalizations	ID, Hosp. start+end

Table 3.1: Table with overview of data sets

3.2 Classification systems for drugs and diseases

There are multiple systems in use for classifications of drugs and diseases worldwide. Following are short descriptions of the systems used in our data sets.

3.2.1 The ATC system

The Anatomical Therapeutic Chemical Classification System [10] is a system for classifying drugs based on the conditions it treats. The system is a five level hierarchy which describes where in the body the drug works, what it treats, in increasing detail, down to the specific chemical compound. This hierarchy enables analyses at various granularity, as drugs that are chemically similar, or that are used to treat the same illness, share up to four out of five levels in the ATC hierarchy. Figure 3.1 shows the ATC-code for Paracetamol, a widely known painkiller, with ATC code N02BE01.

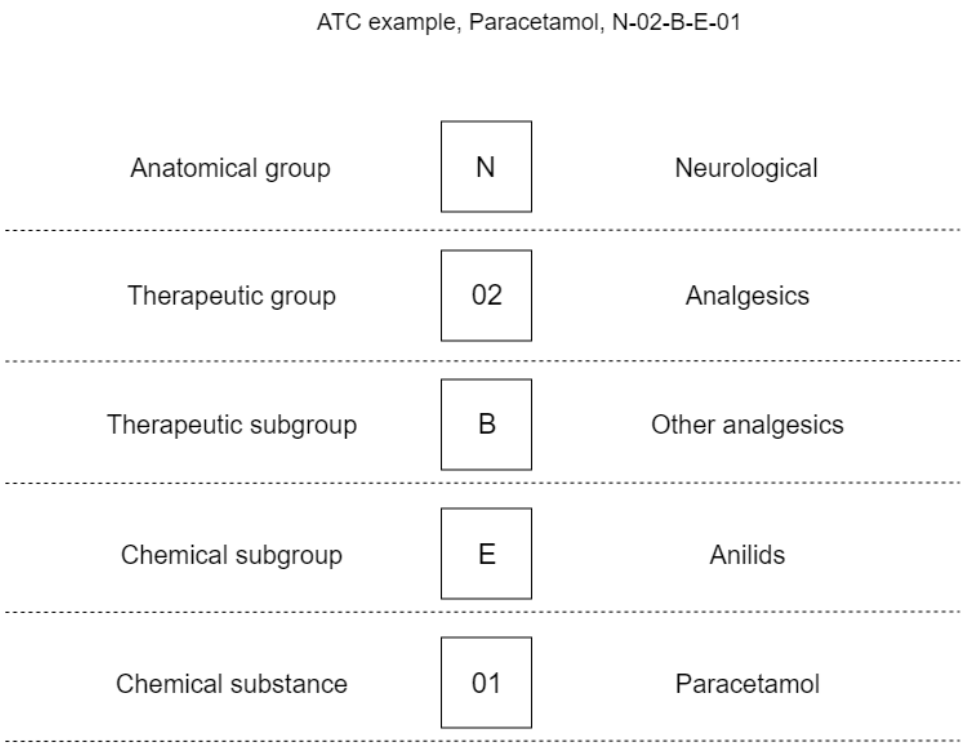

Figure 3.1: ATC Code hierarchy, example with Paracetamol (N02BE01)

3.2.2 ICPC-2 and ICD-10 for classification of diseases

The elders prescription data set uses two systems for disease classification. These are used when medication is paid for by the health care system. However, only one of the two systems are used per prescription, and the prescriber is free to use whichever system he wants to.

Both systems contain most of the same diseases, but use different codes to represent them.

- ICPC-2 [11] (International Classification of Primary Care) is used by primary health care for diagnoses and other health issues. It is commonly used by physicians in Norway.

- ICD-10 [12] is the 10th revision of the International Statistical Classification of Diseases and Related Health Problems. This system is proposed by the World Health Organization, and is one of their latest attempts to create a global classification system. In Norway, this system is widely adopted in hospitals.

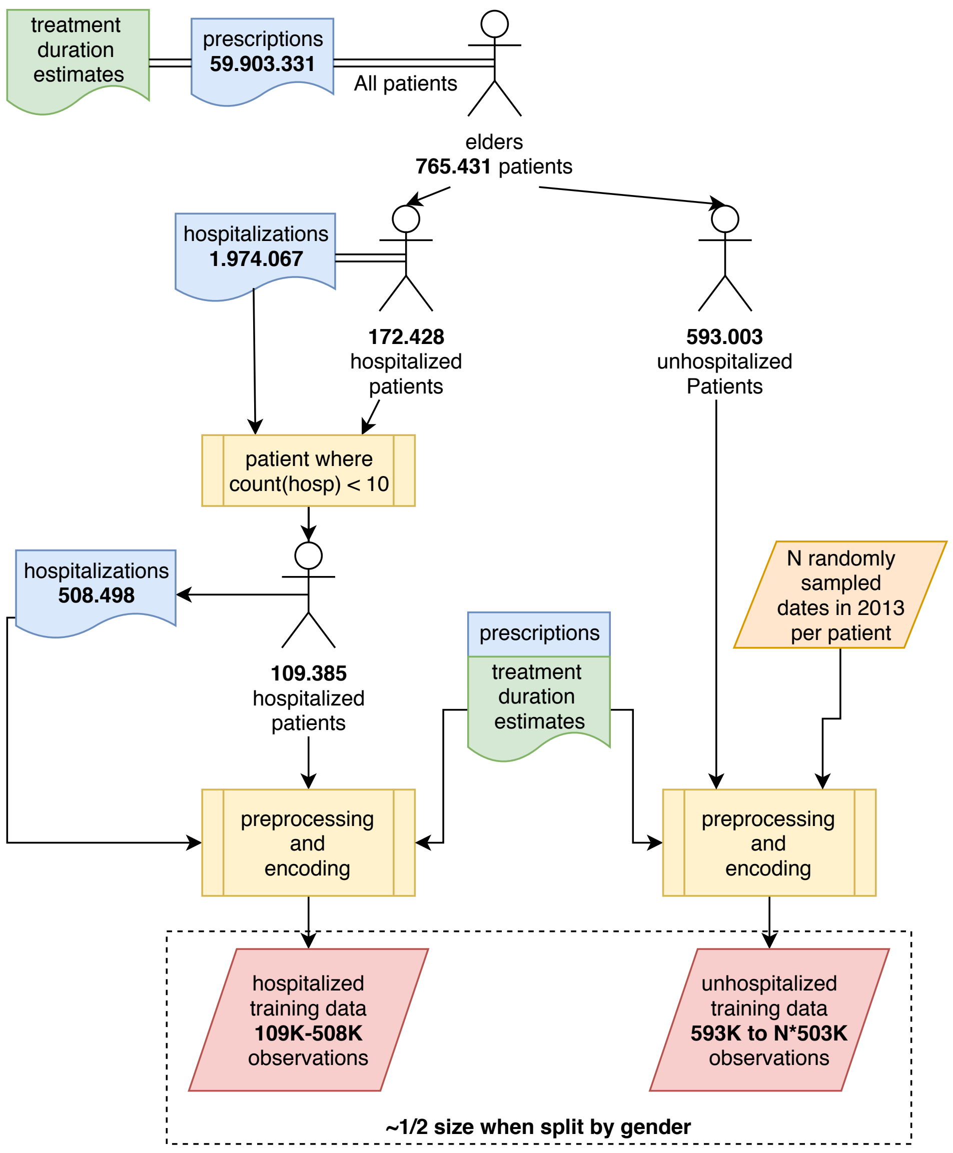
treatment duration estimates
prescriptions 59.903.331
All patients
elders 765.431 patients
hospitalizations 1.974.067
172.428 hospitalized patients
593.003 unhospitalized Patients
patient where count(hosp) < 10
hospitalizations 508.498
109.385 hospitalized patients
prescriptions
treatment duration estimates
N randomly sampled dates in 2013 per patient
preprocessing and encoding
preprocessing and encoding
hospitalized training data 109K-508K observations
unhospitalized training data 593K to N*503K observations
~1/2 size when split by gender

Figure 3.2: Illustration of available data, with eligible subset sizes after cleaning

3.3 NorPD - The Norwegian Prescription Database

We have two data sets which cover drug consumption. The first data set covers most drug consumption of all patients in Norway in a 10-year period. The second covers the drug consumption of all Elders in Norway in a three year period. Both data sets are longitudinal and therefore covers each patient's use of medicines over time. The two data sets cannot be linked with each other directly, as the pseudonymized IDs are different for each of the two sets. This is because they are obtained as part of two different projects. The general population data set contains more than 6x the prescriptions of the Elders data set. However, it has less features, and therefore less detail per prescription, and is not linkable to any other data sets. There are no direct outcomes about the state of the patients, so the potential analyses of this data set is limited. The elders data sets contains less prescriptions, but has more information per prescription. It can also be linked with the other available data sources.

Both data sets are originally in csv-format, and each row contains one prescription. We may describe the information using the ER diagram in Figure 3.3.

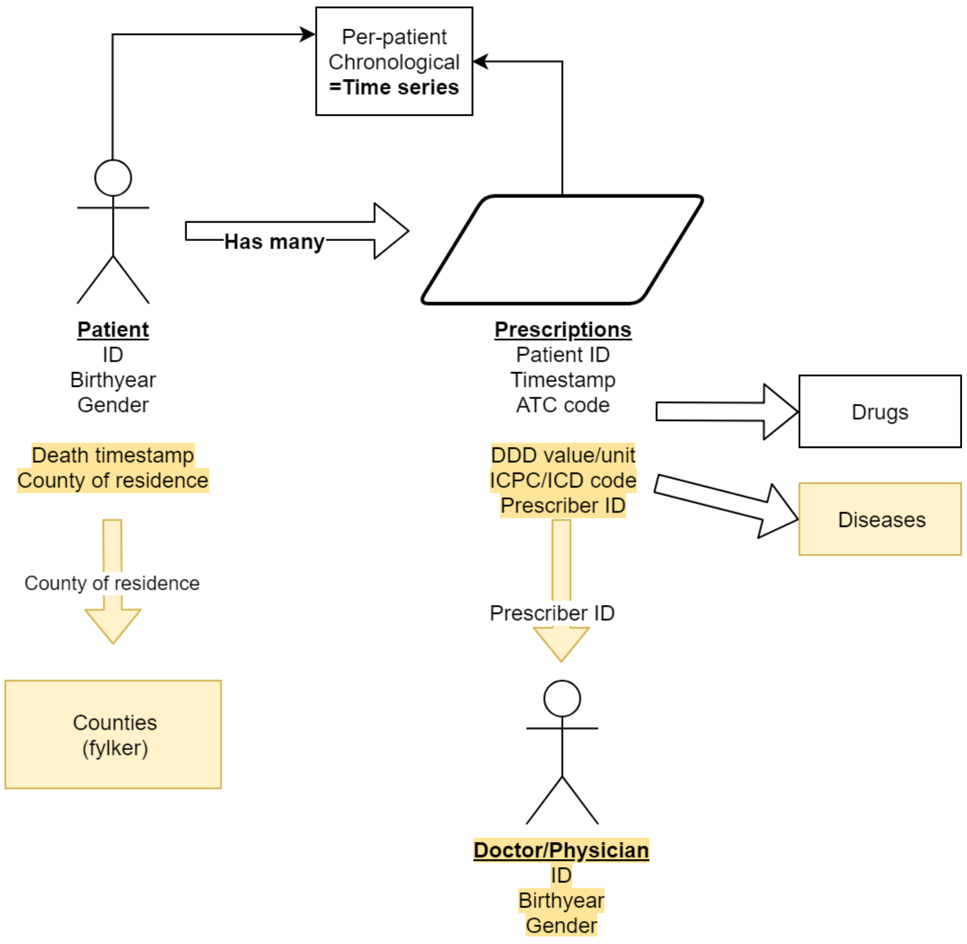

Figure 3.3: ER-diagram of prescription data sets

3.3.1 NorPD - Norwegian prescription register, full population, 2004-2014

This data set contains most prescriptions dispensed at pharmacies in Norway between 2004 and 2014. The data set has 374.9 million prescriptions. It contains 856 unique drugs that have at least one recorded adverse effect in the period. It does not contain drugs that are purchased without prescription, nor drugs used in hospitals.

As shown in our previous work, analyses of all combinations of drugs grow combinatorially. For example, with 856 different drugs, analyses of all combinations of two results in matrices with more than 730.000 different drug combinations, if we include ordering. All combinations of three results in more than 630 million combinations. Creating interpretable and computationally feasible combinatorial analyses may require narrowing of scope, or implementation of novel methods. The structure of the data set is described below.

Structure

Listed are the columns included in the NorPD general population data set. Official Norwegian names are listed in parens.

- Pseudonymized ID (PasientLøpeNr): Pseudonymization instead of deidentification allows us to collect specific patient's prescriptions by searching through the data set.
- Gender of patient (PasientKjønn)
- Birth year of patient (PasientFødtÅr)
- An exception tag for patients without an ID (PasientUtenID): Approximately 1% of medication are prescribed to patients with no ID
- Date of dispensing from pharmacy (UtleveringsDato)
- ATC-Code (ATCKode): Precise classification of the drug

3.3.2 NorPD - elder population, 2012-2014

Its structure is similar to the general population data set. It contains fewer prescriptions, 61.9 million, but has additional variables per prescription about dosage, disease, and variables for the patient including diseases and death. It contains 3 years of prescriptions from patients older than 65 years, between 2012-2014. The additional information allows it to be split into multiple subsets, such as living patients, dead patients and hospitalized patients. This is useful, as it enables creation of labeled data to train supervised learning algorithms with. The data set is obtained as part of a different project than the general population, and thus the patients in the two sets are not linkable with IDs.

Structure

- Pseudonymized ID (PasientLøpeNr)
- Gender of patient (PasientKjønn)
- Birth year of patient (PasientFødtÅr)
- ATC-Code
- Patients county of residence (PasientBostedFylkeNr, -Navn)
- Patients year and month of death, if the patient died between 2013 and 2017 (PasientDødsÅr, PasientDødsMnd)
- Prescribers Pseudonymized ID, age and gender (ForskriverLøpeNr, -FødtÅr, Kjønn)
- Date of dispensing (UtleveringsDato/Diff_UtleveringDato): For persons with no hospital admission in 2013 a simple date is provided. For persons admitted, a relative date is provided with number of days before/after the first hospital admission in 2013 provided. Note that this number may be both positive and negative. This means that a subset of the data must be treated differently with respect to time.
- Amount of drug dispensed (in number of defined daily doses)
- Diagnostic codes for reimbursement (RefusjonKodeICDNr, -ICPCNr): When a prescription is payed for by the health care system, prescribers must provide an indication for use in the form of a ICD-10 or ICPC-2 code. Not all prescriptions are reimbursed so there are missing values in these variables.
- Product name (VareNr, VareNavn)
- ATC DDD value and unit (AtcKodeDDDVerdi, -DDDEnhet): Dosage speci-

fications

3.3.3 NorPD - treatment duration estimates of prescriptions in elder population

This data set contains estimates for the treatment duration of all prescriptions in the NorPD Elders data set, and thus the two sets are directly linkable. However, the timestamp format of unhospitalized patients is different. For unhospitalized patient, the treatment start is defined as the difference in days between the date of prescription, and January 1st, 2013. For patients who were hospitalized, the treatment start variable is the same as the Diff_Utleveringdato. For both sets, the treatment end timestamp is defined as *treatment_start* + *estimated_treatment_time*. The treatment duration estimates are computed by researchers at the institute for pharmacy at UiT, using pharmacoepidemiological rules.

Structure

- Pseudonymized ID (PasientLøpeNr)
- ATC code (ATCKode)
- Treatment start
- Treatment end

3.4 NPR - Norwegian Patient Registry

Our NPR data set contains all hospitalizations in the 2012-2014 period from patients who were hospitalized at least once in 2013. Each row contains a hospitalization, with the ID of the patient, the start timestamp and end time The data is linkable with the NorPD elders data set by the patient's identifier.

Structure

- Pseudonymized ID (PasientLøpeNr)
- ICD-10 code

- ICPC-2 code
- Hospitalization start
- Hospitalization end

3.5 Data ingestion with Spark

To enable fast queries on the data sets, we use Spark to transform the data sets to a Parquet format. We ingest the raw CSV files with Spark SQL's DataFrame API, which parses the header and infers the schema automatically.

Spark SQL defaults the type of each column as string when loading csv files. We make adjustment to some columns. Specifically, we convert dates from string format to unix time, and date diffs and birthyears to integer format. We partition the tables by patient IDs before we write the tables back to disk in Parquet format. , we write the resulting DataFrame to disk in Parquet format. Figure 3.4 visualizes this process.

Queries on a Parquet formatted table is very fast for a number of reasons. First, Parquet is a column-store format which utilizes column compression, which reduces storage footprint, and indexing methods to make it very fast to search from. Queries only loads the columns required into Spark, minimizing the memory footprint and computational complexity. Second, by partitioning by ID we enable fast queries on this field. In a distributed setting it should ensure that queries on a per-patient basis is computed on a per-worker basis, because the partitioning ensures that all data of each respective patient is stored together per table.

Finally, the storage footprint is greatly reduced with Parquet. The elder's prescriptions data set, which storage footprint is 14 GB in csv format, is reduced to 768 MB in Parquet format. The drug treatment duration is 17 MB in Parquet format, down from 760 MB in the raw format. We believe most of this comes down to redundancy in all data sets, which is greatly optimized with column compression.

The reduced storage footprint comes at a cost of increased computational complexity when writing the data to disk. This is negligible in our case, the conversion takes approximately 2 minutes per data set, and only has to be done once. Therefore we believe the up-front computational cost is negligible.

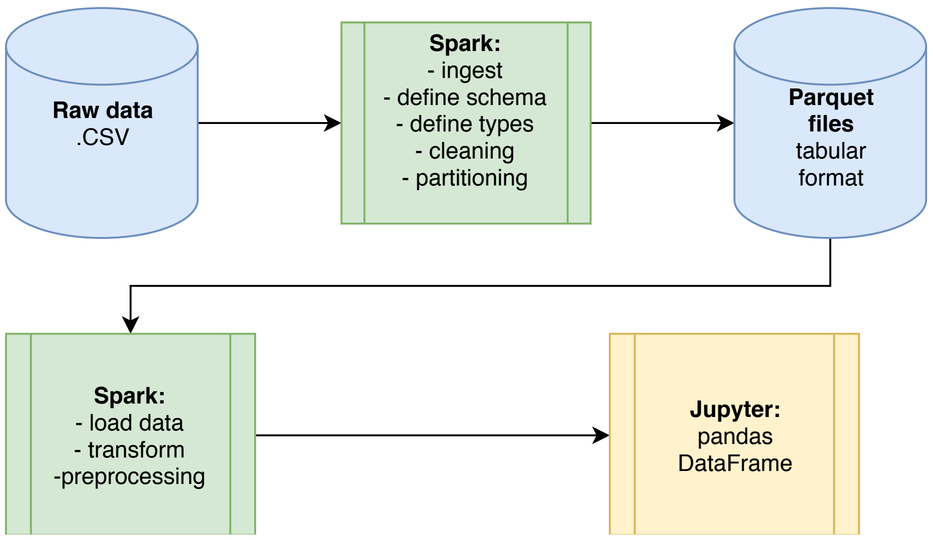

Figure 3.4: Data in csv format ingested by Spark, converted to DataFrame, written to disk in Parquet format

3.6 Analyses that can be done using Spark SQL

3.6.1 Visualizing overrepresented drugs in patients who die

We aim to find out when drugs that are overrepresented in patients who die are consumed relative to their month of death. We therefore implement an analysis of drug consumption in live and dead male patients.

We first have to estimate which drugs that are overrepresented in the dead population. We therefore split the elders data into two subsets, patients who are alive (640.000 patients), and patients who died between 2013 and 2017 (133.000 total).

Compute the relative frequency of each drug per group. For both splits, we count the number of times each drug is prescribed, and also the total number of prescriptions in each respective group (7.5 million for dead patients, 16.2 million for live patients). We normalize the aggregated prescriptions by dividing the count of each drug by the total number of prescriptions for each respective group.

We compare the relative frequencies from the distribution of drug consumption of the two groups on the log scale. We do this by dividing the relative frequencies of dead patients with the frequencies of live patients. We choose the five drugs with the highest relative frequency.

Results

In figure 3.5, we visualize the prescriptions of these drugs in dead patients as a set of 30-day aggregates in a 900 day time frame prior to patient's death. We choose 30 day buckets for two reasons. First, the death date is only defined in year and month, not day, which inherently limits the precision of the timeline. Second, since most displayed drugs occur in low quantities, buckets of smaller size will only make the visualization noisy.

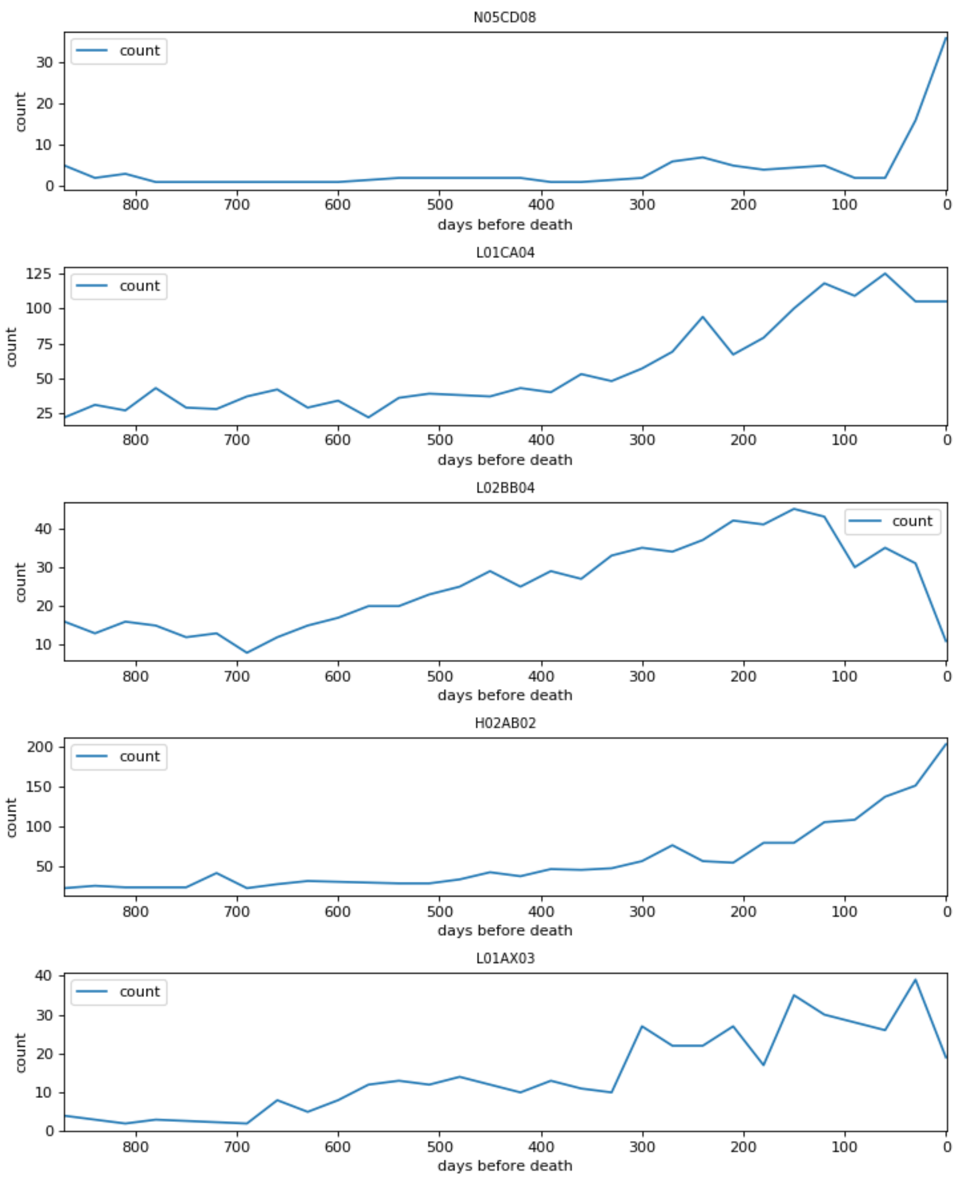

Figure 3.5: Aggregated drug consumption of drugs overrepresented in patients that died, 30-day bins.

We observe that midazolam (N05CD08) is almost exclusively prescribed a very short period prior to death. It is a sedative, and is most likely used for palliation.

L01AX03, L02BB04, L01CA04 are all drugs used in cancer treatment, and we see that the prescriptions tend to occur more frequently when closer to patients death.

Finally, dexamethasone (H02AB02) increases in use when closer to death. This drug is used together with cancer treating drugs, specifically to treat or reduce the severity of adverse effects from the cancer medication, which tend to appear when the treatment is prolonged.

4 Exploring drug use patterns associated with hospitalization

4.1 Introduction

We aim to find drugs that are over- or under-represented in patients that are hospitalized. Our long term goal is to predict adverse drug reactions that cause hospitalization.

We compare drug consumption data for hospitalized and unhospitalized patients.

As a starting point for our research, we design a preprocessing scheme to convert prescription- and hospitalization data from elders to observations in a numerical format, which is usable for modeling. We design two analyses which trains supervised learning models on the observations.

First, we design a comparative analysis to find out how well models can predict hospitalization from our data. We compare the performance of a logistic regression and a neural network in prediction of hospitalization.

Second, we estimate drugs that are over- and under-represented in patients who end up hospitalized and unhospitalized, respectively. We obtain these estimates by applying a bootstrap scheme which computes point estimates and confidence intervals of logistic regression parameters. Since each of the model parameters corresponds to a drug, we interpret the mean magnitudes as a measure of over-representation in the respective groups, and use the confidence intervals to indicate if we have enough data on the respective drug.

4.2 Data preprocessing

To enable supervised learning on our data, we first have to preprocess it. We choose drug consumption as our features, and hospitalization as our outcome. We split the data by gender. This is because the drug consumption of men and women differs; some drugs are exclusively prescribed to one of the two genders.

We further divide the genders into a hospitalized and unhospitalized group. From Chapter 3 we see that Hospitalized patients can be easily identified by selecting all patients in the NorPD Elders data set that use the diff_date field for time stamps.

We use binary labels, and code our unhospitalized group with a 0, and the hospitalized groups with 1. Figure 4.1 displays how our data sets are preprocessed.

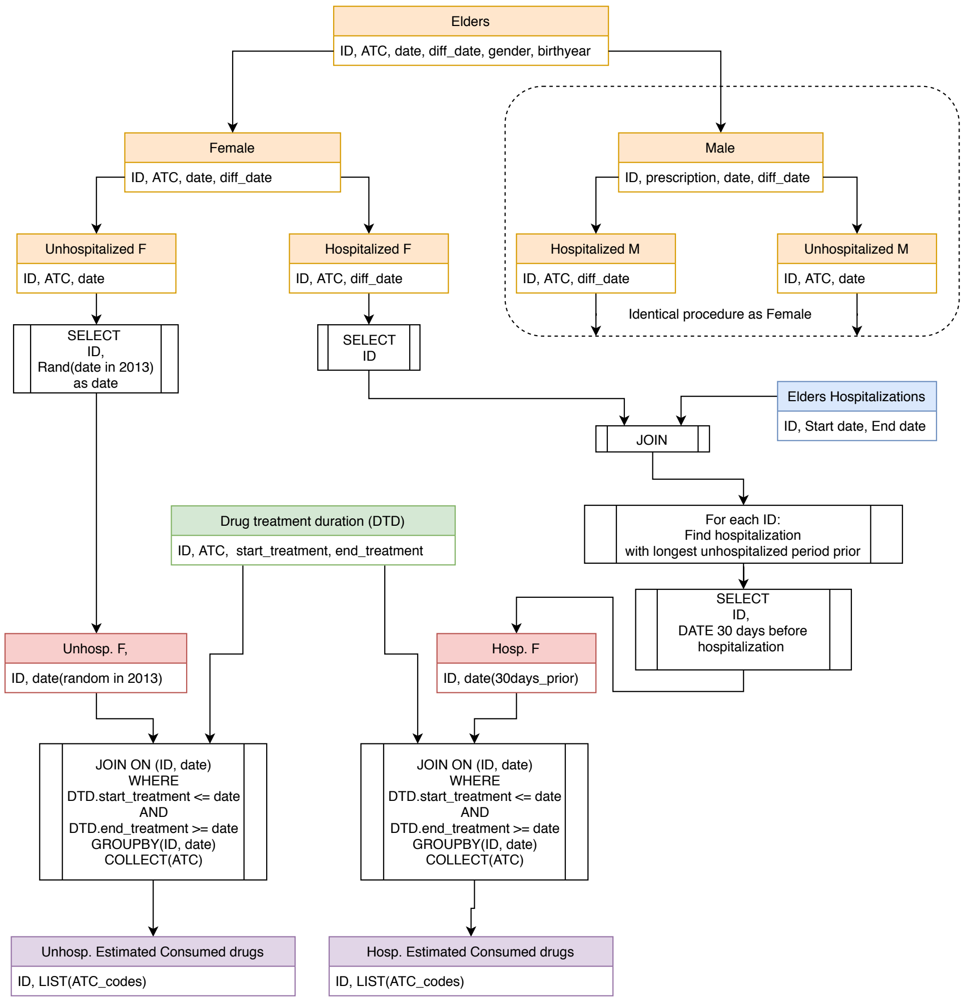
Elders
ID, ATC, date, diff_date, gender, birthyear
Female
ID, ATC, date, diff_date
Male
ID, prescription, date, diff_date
Unhospitalized F
ID, ATC, date
Hospitalized F
ID, ATC, diff_date
Hospitalized M
ID, ATC, diff_date
Unhospitalized M
ID, ATC, date
Identical procedure as Female
SELECT ID, Rand(date in 2013) as date
SELECT ID
Elders Hospitalizations
ID, Start date, End date
JOIN
For each ID: Find hospitalization with longest unhospitalized period prior
Drug treatment duration (DTD)
ID, ATC, start_treatment, end_treatment
SELECT ID, DATE 30 days before hospitalization
Unhosp. F,
ID, date(random in 2013)
Hosp. F
ID, date(30days_prior)
JOIN ON (ID, date) WHERE DTD.start_treatment <= date AND DTD.end_treatment >= date GROUPBY(ID, date) COLLECT(ATC)
JOIN ON (ID, date) WHERE DTD.start_treatment <= date AND DTD.end_treatment >= date GROUPBY(ID, date) COLLECT(ATC)
Unhosp. Estimated Consumed drugs
ID, LIST(ATC_codes)
Hosp. Estimated Consumed drugs
ID, LIST(ATC_codes)

Figure 4.1: Illustration of preprocessing scheme

Data cleaning

Patients with chronic diseases are regularly hospitalized. We need to remove these, because their hospitalizations are usually not caused by drug consumption. As can be seen in Figure 4.2, a majority of patients are hospitalized less than 10 times in the period 2012-2014. We assume that most patients that are hospitalized 10 times or more are likely regularly hospitalized due to a chronic disease. These patient's drug consumption are unlikely to generalize to other patient's drug consumption. We omit these patients, because they are likely to cause our models to overfit. As can be seen in figure 4.3, which shows the number of hospitalizations per bin, we lose a major portion of the total hospitalizations in the data. The resulting set contains 508 498 hospitalizations, 25.7% of the original set of 1 974 067.

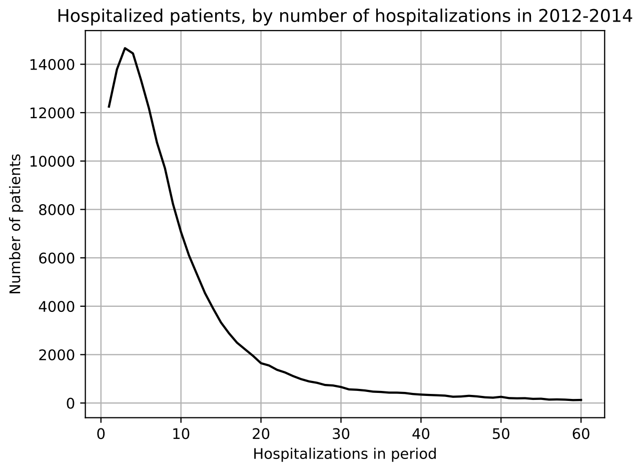

Figure 4.2: Number of patients with N number of hospitalizations, discrete bins

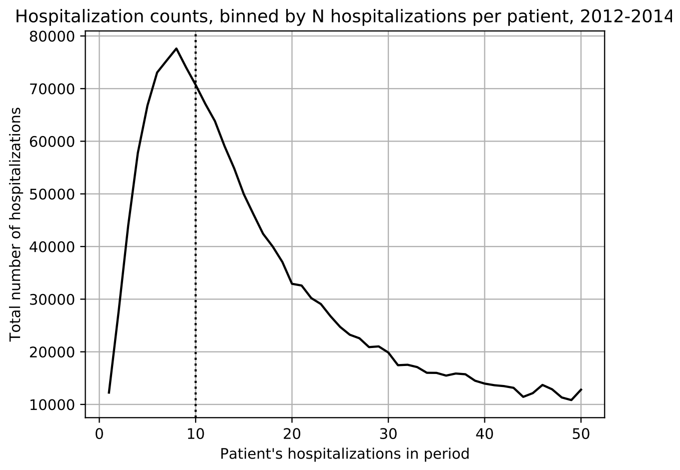

Figure 4.3: Hospitalization counts binned by the number of hospitalizations, discrete bins

ATC levels for dimensionality reduction

The ATC system enables dimensionality reduction in our modeling. This is because the ATC system is a hierarchical drug classification system (as mentioned in chapter 3). A 4-level ATC code describes the therapeutic effect of all 5-level ATC codes with the same preceding levels.

We use a a one-hot encoding on drug consumption - each drug defined by an ATC code - to generate features. This assigns each unique N-level ATC code a binary feature. With 5 levels, this results in more than 900 features in a preprocessed data set. With 4 ATC levels, the number of features is reduced to 160.

Our preprocessed data sets have sparse features. For each observation we can expect only about 4 of the 160 to 900 possible drugs to be consumed per observation. Hence the remaining 97.5% to 99.6% of features respectively are zeroes.

Differences between preprocessing in our analyses

The preprocessing schemes in our analyses have two main differences. First the granularity of ATC codes differ; in the comparative analysis we use 4-level ATC codes, while we use 5-level ATC codes in the bootstrap analysis. Second, the method we use to estimate drug consumption differs. In the comparative analysis we estimate the drugs by examining 30 day windows and encoding the drugs that are prescribed in the window. In the bootstrap analysis, we obtain the drug consumption as point estimates from the treatment duration dataset. The drug treatment data set was made available to us after development of the comparative analysis.

4.3 Logistic Regression vs Neural Network for prediction of hospitalization

We design an analysis where our aim is to determine if prediction of hospitalization is possible from drug consumption data. We compare two models, a logistic regression and a neural network with one hidden layer. We train these models on drug consumption which we estimate by examining 30 day windows in hospitalized and unhospitalized patients. We assess the models performance in the binary inference task on a test set which we derive with a 67%-33% train-test split on the preprocessed data prior to training.

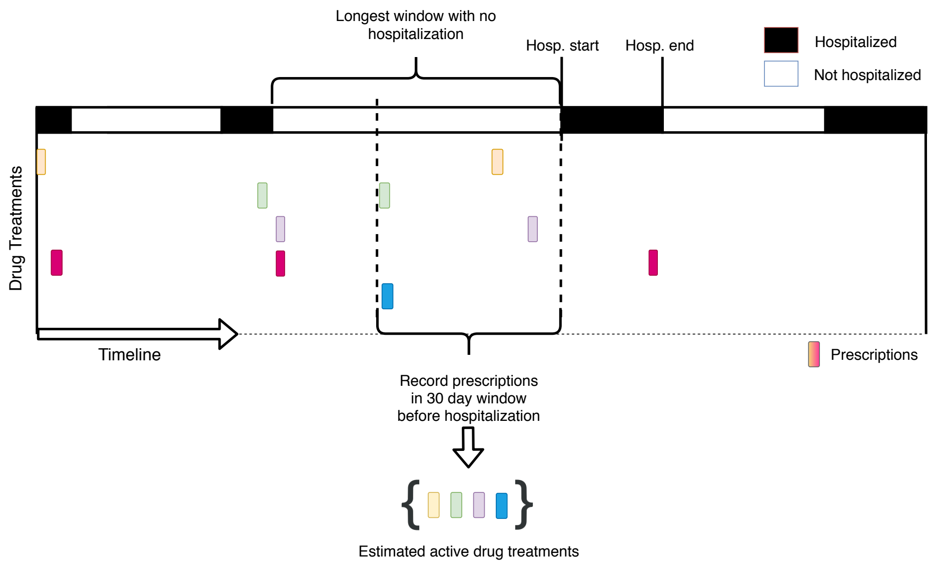
Longest window with no hospitalization
Hosp. start
Hosp. end
Hospitalized
Not hospitalized
Drug Treatments
Timeline
Prescriptions
Record prescriptions in 30 day window before hospitalization
Estimated active drug treatments

Figure 4.4: Drug consumption estimation in comparative analysis

For the people who were hospitalized one or more times we examine 30 day windows immediately before a hospitalization. We code the occurrence of a given drug being prescribed one or more times by a 1, otherwise we code it as a 0. We only select such windows when there is at least 60 days since the last hospitalization. We do this to reduce the likelihood that our sampled dates are correlated strongly with any previous hospitalization. We take up to three of these windows per patient, in descending order by the number of days since the last hospitalization.

We use drug consumption from unhospitalized patients as controls for comparison. We select uniformly at random 30 day windows in 2013 for each patient. We code prescriptions in the same way as above. We sample dates exclusively from 2013 because our data only includes hospitalizations if the patient was hospitalized at least once in 2013. We do this because we know that our unhospitalized group has no hospitalizations in 2013, but they may have hospitalizations in 2012 and 2014 which are not included in the raw data.

We code ATC codes at 4 out of 5 levels. This reduces the number of unique ATC codes to 160, from 900. 4-level ATC retains the function of the drug treatment and chemical group, but loses information about the specific chemical compound.

4.3.1 Logistic Regression model

We model the probability of hospitalization after 30 days, p given that a drug has been prescribed ($x_i \in \{0, 1\}$) as linear on the logit scale

$$\log \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k.$$

The coefficient β_i for drug i represents the change in log odds of hospitalization if drug i is prescribed, all else treated as fixed. In other words, the prescription of drug i changes the odds of hospitalization by a factor of e^{β_i}. Large positive coefficient estimates indicate that drug i strongly correlates with hospitalization. The number of cases and controls we select does not reflect the population frequency of hospitalization, as such the intercept β_0 has no immediate interpretation.

We implement the logistic regression model using scikit-learn's *LogisticRegression* model. We use the liblinear [13] optimizer. The model implements L^2 penalty. We eliminate this penalty by setting the C parameter to 10^{42}. The C parameter is an inverse regularization parameter ($1/\lambda$), thus small C leads to high regularization, which disincentivizes the model from overfitting. An infinitely large C eliminates the regularization, which enables the parameters in the logistic regression to converge unconstrained, but may lead to overfitting.

4.3.2 Neural network model

We design a neural network with a single hidden layer with 50 perceptrons. We use scikit-learn's *MultiLayerPerceptron* model. This model has the same API as the logistic regression model, which makes them simple to compare because it is simple to compute the same metrics on both models. As activation function in the hidden layer, we use the logistic activation function. Our chosen optimizer is Adaptive Moment Estimation [14] (ADAM).

4.3.3 Results

We train both models with the training set (67% split). We compute the receiving operating characteristic (ROC) curve of the models on the test set (33% split). We visualize this curve for the respective models in Figure 4.5.

As can be seen, both models achieve a near-identical area under the curve on the test set. The neural network achieves an AUC score of 0.771, while the logistic regression model achieves an AUC score of 0.758. The neural network achieves slightly higher true positive rate than the logistic regression when false positive rates are low.

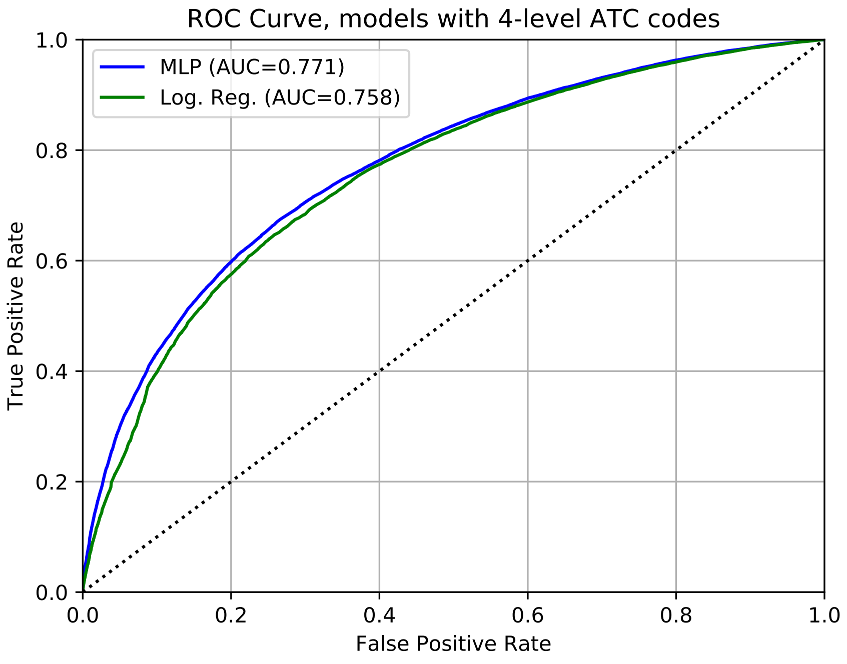

Figure 4.5: ROC curve of Multi Layer Perceptron (Neural Net) and logistic regression

As can be seen in the calibration curve (Figure 4.6), both models appear to be well-calibrated. A perfectly calibrated model has probabilistic output. This means that if the model predicts an 80% probability of hospitalization for a large number of observations, 80% of the predictions are correct.

Neural networks are composed from layers of stacked linear models with non-linear activation functions between them. Neural nets can in model complex interactions between drugs from the data, but it comes at the cost of interpretability; it becomes increasingly difficult to understand its inference process when the number of layers increases, and the network also grows more prone to overfitting.

On the other hand, in a logistic regression, each parameter directly corresponds to consumption of a single drug. This enables direct interpretation of how each drug affects inference.

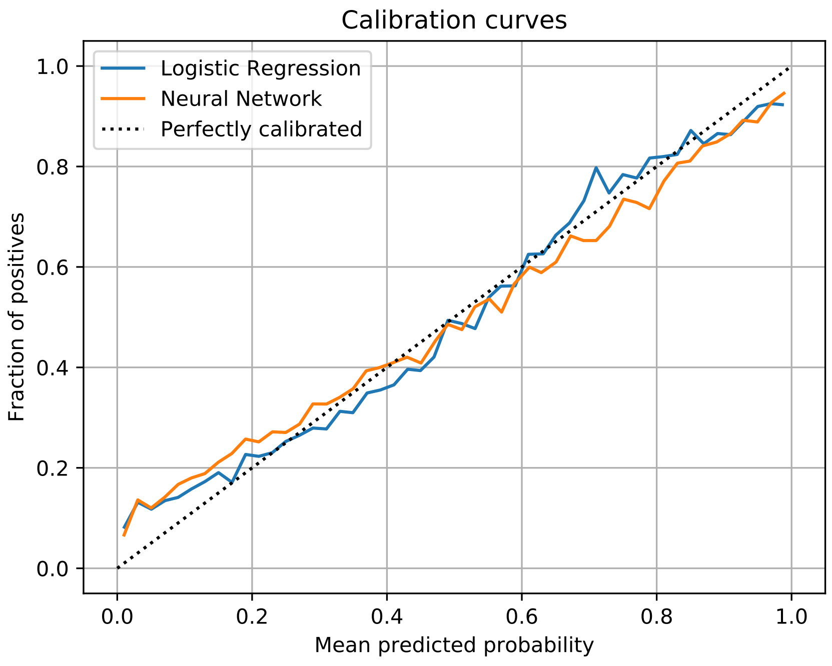

Figure 4.6: Calibration curve of Multi Layer Perceptron (Neural Net) and logistic regression

The neural network has approximately 8000 parameters, 50 times the parameters of the 160 parameters in the logistic regression models. When compared to the logistic regression, this has two major drawbacks. First, it is much more difficult to interpret the decision making process of the neural network than the logistic regression model. Second, the model is much more computationally complex to train. The neural network takes 400 seconds to train on CPU, which is 65 times longer than the logistic regression training time of 6.1 seconds.

The neural network shows signs of overfitting. It has a training set AUC of 0.822, a difference of 0.05 from its test set AUC. On the other hand, the logistic regression achieves almost identical AUC between the training and test set, with an AUC of 0.762 on the training set, a difference of 0.004.

Interpretability is an important topic in all research. Especially in fields related to medicine, researchers are often willing to trade off performance for interpretability. It only makes sense; many people won't let an analysis system which reveals nothing about what it derived its decision from, be used in a safety-critical or health-critical application.

We believe logistic regressions provide a more useful result than the Neural Network because of this. The logistic regression is interpretable with regards to our data, while the Neural Net has no immediate interpretation. Also, with the performance metrics being almost identical, and the training time of the neural network, there seems to be little application for this type of model with the data we have now.

Our results indicate that there may be some inherent problem in our data or preprocessing scheme. It is possible that the performance of our models is limited by overlapping feature distributions in the hospitalized and unhospitalized groups, and that more accurate prediction cannot be made. However, we believe it is more likely that out current cleaning - where we omit patients that have less than 10 hospitalizations - fails to remove many hospitalizations that are definitely not caused by drug consumption.

To improve the performance metrics of our data modeling, we believe the data cleaning in the hospitalized patient group should be improved. To do this, we suggest omitting hospitalizations that are non-preventable by drug treatment. However, determining which hospitalizations to omit will require domain expertise and time to implement.

4.3.4 Conclusion

Our models perform equally well in prediction, and they appear to be well-calibrated, which means that they have some probabilistic properties. While the ROC curve shows that models dont perform badly in prediction, they don't perform great either. We cant determine if we can achieve better prediction of drug consumption, but we believe that improved preprocessing of our hospitalized group can improve our models prediction performance.

4.4 Boostrapped logistic regression

Following the results from the comparative analysis, we design an analysis to detect drugs that are over-represented in hospitalized patients. To achieve this, we apply a bootstrap scheme where we replicate logistic regressions. This scheme also allows us to estimate drugs that we lack observations on. We use a similar preprocessing as in the comparative analysis, and use unhospitalized patients as a control.

4.4.1 Data differences from the comparative analysis

The data used in this analysis is derived using the same method as the comparative analysis, with two differences.

First, we use 5-level ATC codes, instead of 4-level. This results in 900 features, compared to the 160 features in the data in the previous analysis.

Second, we obtain drug consumption point estimates by performing a look up in the active drug treatments dataset for each we sampled date per patient. In patients who are hospitalized, we obtain these point estimates 30 days prior to a hospitalization, with the same constraints as in the previous analysis. In unhospitalized patients, we obtain the point estimates from dates sampled uniformly at random in 2013.

The point estimates should be more accurate than the estimates in the comparative analysis, because the treatment durations we use for our drug consumption sampling are computed using pharmacoepidemiological rules. These drug consumption estimates are thus more likely to include drugs that are prescribed for longer periods than 30 days at a time, which the method in the previous analysis can't reliably estimate.

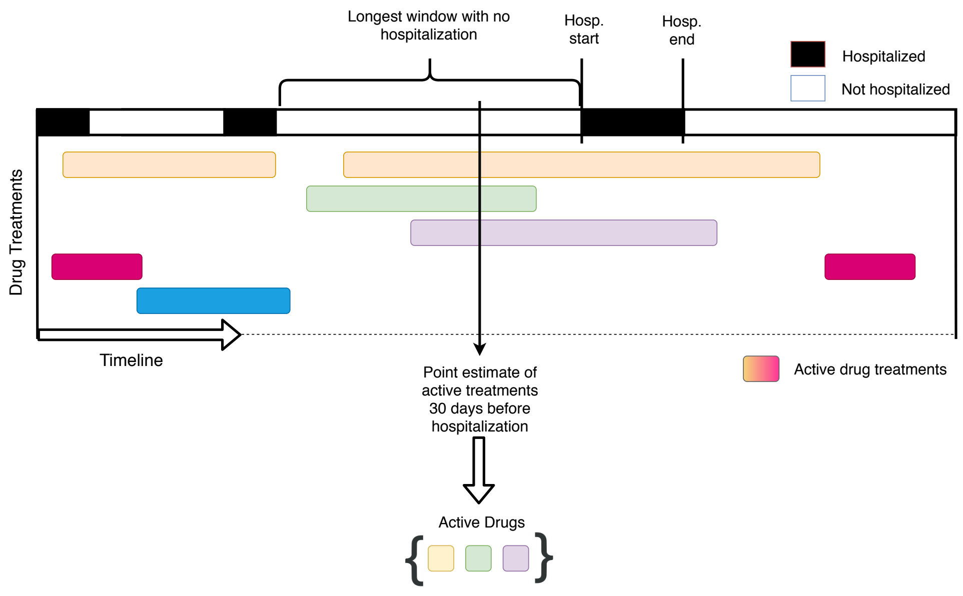

Figure 4.7: Drug consumption estimation in analysis 2

4.4.2 Methodology

We implement a bootstrap method to compute point estimates and distributions of a logistic regression model. To do this, we simulate 2000 data sets from our population by drawing observations from both populations at random with replacement. For each bootstrapped data set we perform a train-test split of 66%-33% and fit a logistic regression to it.

We compute the point estimate of each parameter β_i, in the bootstrapped logistic regressions ($n \in \{1, 2, ..., 2000\}$) by

$$\mu_i = \frac{\sum_{n=1}^{2000} (\beta_i^n)}{2000}$$

and the standard deviation

$$\sigma_i = \sqrt{\frac{\sum_{n=1}^{2000} (\beta_i^n - \mu_i)^2}{2000}}$$

Since each parameter β_i directly corresponds with a drug i we use the distribution we compute to detect drugs that are over-represented with respect to hospitalization.

The distribution we obtain enables us to estimate which parameters that have enough data to be considered reliable in future data modeling, and which do not.

Real \ Predicted	True	False	Sum
True	5.92%	20.73%	26.65%
False	3.04%	70.30%	73.45%
Sum	8.96%	91.04%	100.0%

Table 4.1: Confusion matrix of a bootstrapped logistic regression model

4.4.3 Implementation

The logistic regression model scikit-learn's *LogisticRegression*. The C parameter and training scheme we use are equivalent to the logistic regression model in Analysis 1. Since 5 levels of ATC codes are used, it has 900 parameters instead of 160.

4.4.4 Results

Confusion matrix

We compute the confusion matrix (Table 4.1) of a logistic regression trained with our bootstrap method. The test set for this specific model contained 88005 observations, of which 23456 (26.65%) were hospitalized, and 64549 (73.45%) were not hospitalized. As can be seen, the model achieved a 66% true positive rate (5206 observations, 5.92% of total), and a 77.2% true negative rate (61868 observations, 70.3% of total). This shows that the model is slightly better at predicting negative samples than positives.

Visualization of the parameter distributions

In Figure 4.8, we observe that many parameters have high variance. This indicates that the corresponding drugs occur infrequently, and the logistic regression is unable to learn stable parameter estimates for them.

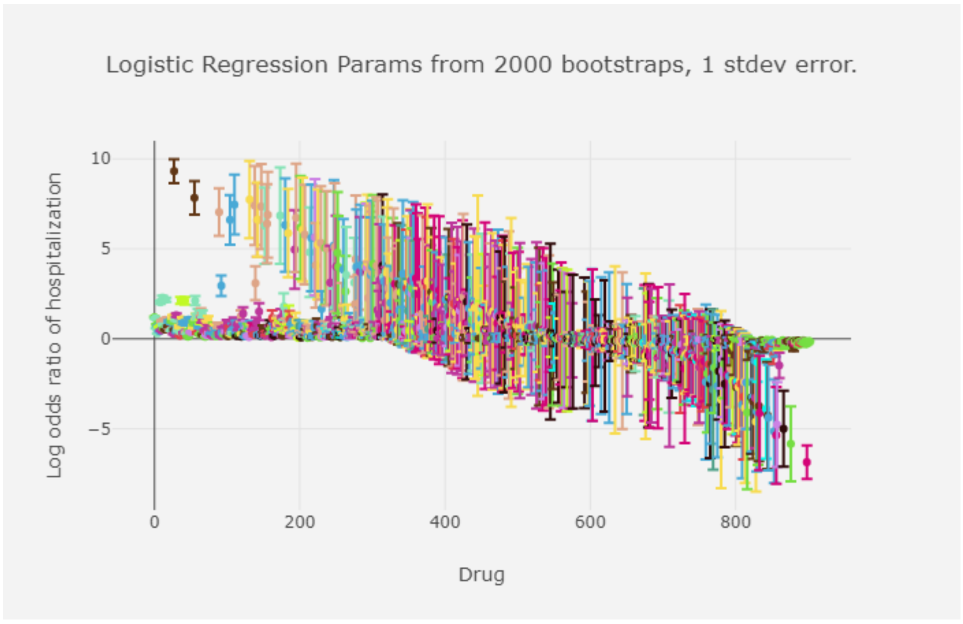

Figure 4.8: All bootstrapped logistic regressions parameter distributions by relevance

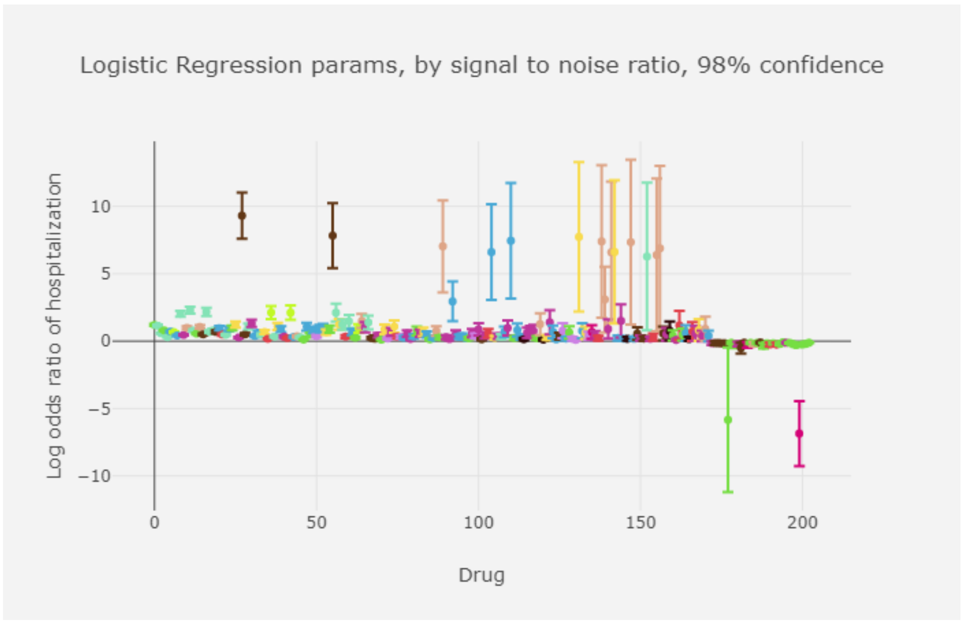

Figure 4.9: Filtered bootstrap logistic regression

In a logistic regression model, 0 log odds means there is an equal probability of each respective outcome. We assume that parameters which 98% confidence interval crosses the line is unlikely to be useful to do inference with using our available data. We therefore omit all respective parameters which 98% confidence interval crosses the 0 log odds (50% probability) line from the visualization. This result is a list of 200 drugs (Figure 4.9).

Conclusion

We have identified a list of drugs associated with hospitalization. We believe this list can be used as a starting point for further analyses.

The large number of parameters with high variance in 4.8 confirms our suspicion that many drugs don't occur frequently enough to enable accurate estimates of their true correlation with hospitalization. To obtain more accurate parameter estimates for these parameters, we believe more data is required.

5 Performance Evaluation

To find out how our system scales, and how we can improve this scalability, we evaluate the runtime performance of our bootstrap algorithm.

5.1 Hardware

Our system and analyses were run on a single machine with the following specs:

- Intel Xeon E3-1275 4C/8T @ 3.80 GHZ
- 64 GB 2400MHZ ECC RAM
- 1x Nvidia GTX 1080 Ti
- 500 GB SSD (500 MB/s Read, 500 MB/s Write)

5.2 Bootstrap methods

The major drawback of our bootstrap analysis is computational complexity. Since point estimates and distributions are obtained by repeated model training on many slightly different datasets, the complexity scales linearly with the number of bootstrap samples we require.

In our case, we performed 2000 iterations of logistic regression on bootstrapped data with males. Figure 5.1 displays the running time of each iteration in the analysis with mean and error lines. The bootstrap iterations took 15 hours and 12 minutes in total on a single machine.

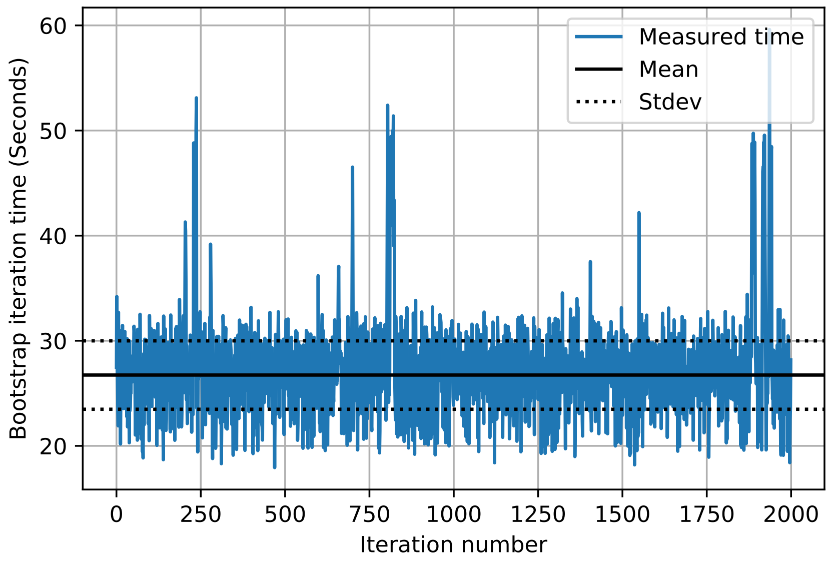

Figure 5.1: Graph showing runtime of all 2000 bootstrap iterations

Figure 5.2 displays the convergence of the bootstrapped mean of every parameter as training commences, with the final mean of each parameter subtracted from every respective data point. We observe that the means stabilize as the number of bootstrap samples increase, and that they are fairly stable after 2000 iterations. This indicates that we probably have enough bootstrap resamples. However, the graph reveals nothing about whether the data is sufficient for accurate estimates (We assume many drugs lack data, as we talked about with Figure 4.8). From the convergence graph, we only know that we have eliminated most of the variance caused by not utilizing the n^n possible permutations of the data set.

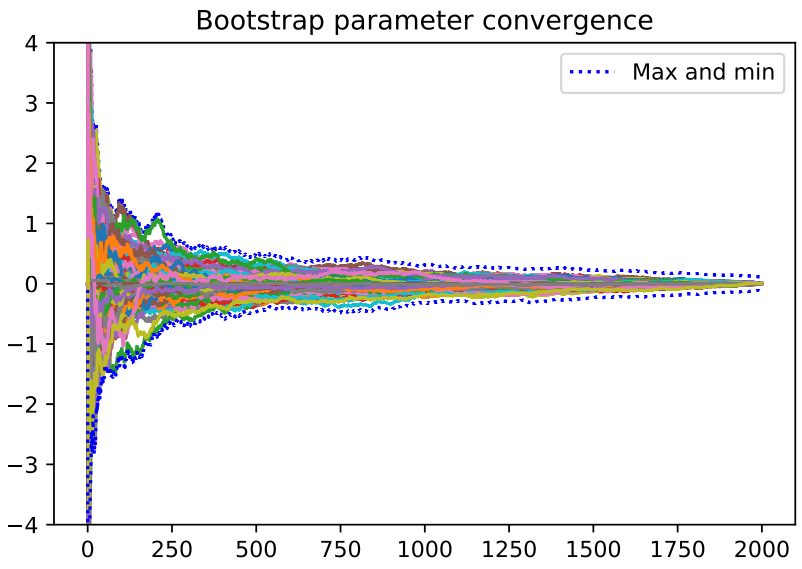

Figure 5.2: Bootstrap convergence of the parameter means. Each data point has the respective parameter mean of 2000 bootstraps subtracted from it.

5.2.1 Reducing the bootstrap analysis time

An advantage of the bootstrap is that it is embarassingly parallel; each bootstrap resampling procedure can be independently computed. Therefore, to reduce the total analysis time on a single machine, we suggest utilizing thread parallelism and distribution.

The data resampling and preprocessing is already partially implemented with Spark. To enable Spark to scale this whole procedure we still need to implement the remainder of the preprocessing scheme in Spark.

To reduce the total training time of the logistic regressions we suggest two improvements.

First, we can improve the throughput on a single machine by training models in parallel on each CPU core. To achieve this with scikit-learn, we suggest using the Joblib Python library, because it requires little work to apply. With four models training in parallel, this should increase the throughput of model training by four. Since approximately 75% of our analysis time was spent training logistic regressions (20/27 seconds), we expect the analysis time to be more than halved. We expect this because the throughput of models that are trained increases four-fold (one model per 20 seconds to 4 models per 20 seconds) while the resampling time per data set remains the same (7s/model*4=28s). This results in an estimated 48 seconds to train four models, and 24 000 seconds (48s*500) to train 2000 models, which is approximately to 6 hours and 40 minutes.

To distribute our bootstrap analysis with logistic regression models, we suggest two solutions. First, the logistic regression training scheme can be implemented with Apache Spark's MLLib. This will require a complete reimplementation of our bootstrap analysis, and we still need to investigate MLLib can be used to achieve it. A second option is to distribute the current bootstrap algorithm as a Python script that to each worker node in the same cluster that Spark runs in, and let each worker node compute a number of bootstrap resamples.

/6 Conclusion

We have designed a system which enables pharmacoepidemiological data exploration. It uses Apache Spark to enable analyses to scale to drug consumption and hospitalization data in 700 000 Norwegian elders, with 60 million prescriptions and 1.9 million hospitalizations in a three-year period. We believe it will scale to data even larger than the sources we have had available.

We demonstrated the usability of our system by implementing two machine learning methods to explore the correlation between drug consumption and hospitalization.

We implemented two models, a logistic regression and a neural network. The models achieved near-identical performance, and both were well-calibrated. The results suggest that there are detectable signals in the data. But we need more data and improved data cleaning in order to be able to improve the prediction metrics

We estimated drugs that are over- and underrepresented in the hospitalized group, using bootstrapping with linear regression on labeled data. The result is a list of the drugs that are most likely over-represented in the hospitalized and unhospitalized group, respectively. The drugs are prioritized based on their signal to noise ratio (μ_i/σ_i). We also found that a large number of drugs that had high variance, and these were therefore omitted since the results suggests that these drugs don't occur often enough in either of our populations. More data is necessary to get more accurate estimates for these drugs.

6.1 Future work

We believe that our system can help pharmacoepidemiologic research in the future. The preprocessing can be further refined by improving data cleaning procedures. Furthermore, new features such as disease classifications can be added to enable machine learning models to learn more complex patterns between drug consumption and outcomes.

We believe our system and analyses will scale to larger data sets of similar type. Spark is the component that makes this possible. To scale machine learning algorithms, we have shown that scikit-learn can be used with JobLib to train multiple models in parallel on a single machine. To scale machine learning analyses to very large data, we suggest distribution with Spark for simple models, and Tensorflow for complex models because it enables distribution and GPU acceleration.

The current iteration of our system uses spylon-kernel. It is not actively maintained. We suggest adopting an alternative kernel in the future. Apache Toree [15] is an incubating Apache project which provides a kernel with the same functionality as spylon-kernel, but with added R and SQL support. Migrating our notebooks to this environment should be simple, since all code is generic.

For cloud based notebook alternatives we suggest using Databricks. This service provides notebook-style environment with managed Spark, much like the system we have composed, but in the cloud. The service sees continous improvements, and has seen growing use in industry since its inception. We believe migration to this service will require the same amount of work as migration to Apache Toree.

Bibliography

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, pp. 10–10, 07 2010.

[2] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," vol. 51, pp. 137–150, 01 2004.

[3] "Hadoop mapreduce." `http://hadoop.apache.org/`. Accessed: 2019-05-31.

[4] "Evaluation metrics - rdd-based api." `https://spark.apache.org/docs/2.1.0/mllib-evaluation-metrics.html`. Accessed: 2019-05-31.

[5] "Apache parquet." `https://parquet.apache.org/`. Accessed: 2019-05-24.

[6] "Project jupyter." `https://jupyter.org/`. Accessed: 2019-05-31.

[7] "Apache spark™ - unified analytics engine for big data." `https://spark.apache.org/`. Accessed: 2019-05-31.

[8] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, *et al.*, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pp. 1383–1394, ACM, 2015.

[9] "Valassis-digital-media/spylon-kernel." `https://github.com/Valassis-Digital-Media/spylon-kernel`. Accessed: 2019-05-31.

[10] "Whocc - atc/ddd index." `https://www.whocc.no/atc_ddd_index/`. Accessed: 2019-05-24.

[11] Wikipedia contributors, "International classification of primary care — Wikipedia, the free encyclopedia," 2018. [Online; accessed 1-June-2019].

[12] Wikipedia contributors, “Icd-10 — Wikipedia, the free encyclopedia,” 2019. [Online; accessed 1-June-2019].

[13] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, “Liblinear: A library for large linear classification,” *Journal of machine learning research*, vol. 9, no. Aug, pp. 1871–1874, 2008.

[14] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *International Conference on Learning Representations*, 12 2014.

[15] “Apache toree.” `https://github.com/apache/incubator-toree`. Accessed: 2019-05-31.

A Supplementary material

The supplementary material and the repository with notebooks and code will be made public some time in the future. Currently they are only available by request because it may contain some sensitive information.

A.1 Notebooks

Following are a description of the notebooks included in which contains the methods we used to ingest csv data and convert it to Parquet on disk. All notebooks are in HTML format, thus they are not executable. Note that the syntax highlighting is wrong in many cells. This is a result of the HTML digest. The executable notebooks have functioning syntax highlighting. Our visualizations are included in these analysis notebooks.

- *notebook_1*: Contains the code required to ingest the general population prescription data set from NorPD.
- *notebook_2*: Contains the code required to ingest the elders prescription data set from NorPD.
- *notebook_3*: Contains the code required to ingest the hospitalization data of elders from the NPR data set.

- *notebook_4*: Contains comparison of distributions of drug consumption between live and dead patients obtained by counting occurrence of drugs. Produces a dataframe with drugs sorted by their estimated overrepresentation.

- *notebook_5*: Visualizes consumption of the top 5 overrepresented drugs (obtained in *notebook_4*) in dead patients relative to their death.

- *notebook_6*: Data selection in the comparative analysis.

- *notebook_7*: Preprocessing and modeling in the comparative analysis.

- *notebook_8*: Data selection for the bootstrap analysis. Also contains conversion of the drug treatment data set to Parquet.

- *notebook_9*: Preprocessing and bootstrap analysis.

- *notebook_10*: Visualizations from bootstrap analysis. Contains interactive plotly visualizations of Figures [4.8, 4.9] near the bottom in the notebook.