www.QualiMaster.eu

QualiMaster

A configurable real-time Data Processing Infrastructure
mastering autonomous Quality Adaptation

Grant Agreement No. 619525

Deliverable 2.2

Work-package	WP2: Scalable, Quality-aware Data Processing Methods
Deliverable	2.2: Scalable, Quality-aware Data Processing Algorithms V1
Deliverable Leader	LUH
Quality Assessor	Apostolos Dollas (TSI)
Estimation of PM spent	28
Dissemination level	PU
Delivery date in Annex I	31.07.2015
Actual delivery date	31.07.2015
Revisions	3
Status	Final
Keywords:	Algorithms, Stream Processing, Evaluation

List of Authors

Partner Acronym	Authors
LUH	Sergiu Chelaru, Christoph Hube, Miroslav Shaltev, Patrick Siehndel, Sergej Zerr
SUH	Cui Qin, Holger Eichelberger
TSI	Ekaterini Ioannou, Minos Garofalakis, Odysseas Papapetrou, Nick Pavlakis
MAX	Maciej Kurek, Steve Hutt
SPRING	Holger Arndt, Stefan Burkard, Walter Colitti

Contents

Executive summary

This deliverable is the second deliverable of WP2. It reports on the progress of WP2 within the first half of the second year of the QualiMaster project and it is concentrating on describing developing and evaluation of the first algorithmic components for real time processing and analysis of data streams within processing pipelines. In the perspective we are talking about so called “sandbox” where the expert will be able to choose and combine the algorithms into pipelines. The selection of the algorithms for the components was done based on requirements in WP1 (Requirement Analysis and Use Case Definitions) derived from collaboration with industrial members of the consortium. Another selection criteria were technical efficiency and scalability of the algorithms as well as comments from the advisory board on ongoing work. Special focus lies on conducting financial (systemic) risk analysis using real-time information from financial market and social streams. The list of the developed algorithm families includes correlation matrix estimators for computing correlation between market players based on their trading information. Here two algorithms were implemented, namely Hayashi-Yoshida Correlation Estimator and Mutual Information. We also consider how the correlation indexing can be distributed and the resulting correlation graph can be queried by experts. We also continued the work on efficient sentiment analysis for social messages where we are developing a pre-filtering component and introduce the hashtag based sentiment estimation approach. We continue our work on event detection and consider two algorithms, namely the “Moving Average” and the “Kleinberg Event Detection”. Finally we introduce a component for extracting social networks between the market players based on their co-mentions in social media.

We started to implementation and first evaluation of the components within first real-time processing pipeline of QualiMaster based on Apache Storm topologies. We took into account the comments from the project review in March 2015. Particularly, social web components are currently designed to process general types of messages such as tweets and news articles from wide range of sources such that the processing is not focused on Twitter only. Furthermore, where applicable, we extended the evaluation strategy for the components and included additional quality measures such as novelty and diversity.

1 Introduction (All Partners)

In this Deliverable we describe design and implementation details of the first set of components encapsulating scalable real-time algorithms and data structures for realizing (systemic) risk analysis. This includes algorithms focused on efficient and scalable processing for sketching, aggregating, correlating or predicting the evolution of several data streams such as financial data, news or Twitter messages. We will describe domain specific characteristics of the data processing and evaluate those components with respect to efficiency, quality, scalability and, where applicable, to other quality measures such as novelty and diversity as suggested from the advisory board. Additional comparison of algorithms within algorithm families will be conducted.

The functionality based selection of the algorithm families is derived by the use cases and user requirements defined in WP1 (as documented in D1.1, D1.2 and D2.2). However, the decision on the specific implementation is mainly based on whether the implementation of the algorithms can scale up with the volume and arrival rate of data in real-time. This is an initial set of components and it can be extended during the course of the project. The different use cases, which are outlined in D1.2, require different quality and granularity of data processing. Depending on the goals, data load peak times are handled differently, regarding filtering of information. Furthermore the use cases differ in their requirements depending on the time resolution of (systemic) risk analysis cycles, which may influence the processing requirements.

1.1 Relation to other Deliverables

For the selection and implementation of the components, we base on use cases presented in D2.2 based on requirements identified in WP1 and under consideration of the description from deliverables D1.1 and D1.2. Some of the considered algorithms are being already translated to reconfigurable hardware in WP3, which are then described in D3.1. Implemented components follow adaptable schema developed in WP4 and described in D4.2. Finally, the deployment, integration and evaluation of the developed algorithms are being performed, when already possible, on the QualiMaster infrastructure that is provided by WP5 and described in deliverables D5.1 and D5.2. Requirements and descriptions from D1.2 can be summarized as follows:

- Key actors of the QualiMaster infrastructure are defined as application users, infrastructure users and component providers.
- Application use cases are defined for Institutional Financial Clients, as well as for Regulatory Bodies.
- Data and data stream requirements, requirements for supported algorithms and platform quality requirements are defined.
- System requirements and use cases are detailed for the following actors: Pipeline Designer, Adaptation Manager, Platform Administrator and Component Providers.

Addressing Comments from First Year Review and Advisory Board

For current experiments we used mainly social web data we collected as discussed in D2.1 and financial data provided by Spring API (particular sets are presented more detailed in corresponding sections). Additionally, we took into account the comments from the project review in March 2015. Particularly, social web components are currently designed to process arbitrary types of messages such as tweets and news articles from wide range of sources such that the processing is not focused on Twitter only. Furthermore, where applicable, we extended the evaluation strategy for the components and included additional quality measures such as novelty and diversity as proposed by reviewers.

1.2 Outline of the Deliverable

The deliverable is structured as follows. First, Section 2(Components for Processing Financial Data Streams) describes components for processing of general financial streams including matrix correlation, distributed similarity indexing and network querying. Next, Section 3(Components for Social Web Data Analysis) introduces components for social web stream analysis. These include prefiltering and sentiment estimation of messages as well as expert modeling component, followed by event detection and social network extraction. Finally, Section 4 summarizes and concludes this deliverable with insights on future work towards the next deliverable, which is due in July 2016.

2 Components for Processing Financial Data Streams

The primary goal of the QualiMaster project is financial risk analysis, with emphasis on the systemic risk. We describe the work progress on processing of financial data streams. In particular, we continue our work on various implementations of the correlation matrix estimators (Section 2.1), on the distributed similarity indexing for minimizing existing limitations (Section 2.2) and we started work on the market player data encoding network (Section 2.3). The network is crucial for systemic risk detection and is based on in-between market player correlation estimation.

2.1 Correlation matrix estimators

There are various measures for correlation computation and this family aims at equipping the QualiMaster infrastructure with representative options. Table 1 provides an illustration of the correlation estimators that we plan to incorporate by the end of the project. A symmetrical dependency does not infer a directional relationship, i.e., one stock price drives another stock price. It only says that there is some sort of relationship between two market players, for example there exists a linear correlation of prices. A symmetrical dependency is usually defined as causality, i.e., one stock price is dependent (not solely) on the past prices of a different stock. If the measure can capture directionality a directed graph can be constructed.

The symmetrical dependency measures are cheaper to compute and the directional measures more computationally intensive since they require computation of directionality in both directions. Taking into account non-linear relationships between two data vectors also imposes additional computation costs. Despite the computation costs, non-linearity might be crucial for the identification of systemic risk and thus it is important to also include such measures with the QualiMaster infrastructure.

	Symmetric	**Asymmetric**
Linear	Hayashi-Yoshida linear correlation coefficient	Hayashi-Yoshida linear cross-correlation coefficient
Non -linear	Mutual Information	Transfer Entropy

Table 1: The measures in the family of correlation matrix estimator.

The following paragraphs provide an overview and discuss the status of the measures from the family of correlation matrix estimator that are currently incorporated in the QualiMaster infrastructure.

2.1.1 Hayashi-Yoshida estimator

This is a linear correlation coefficient for measuring the linear relationship between two vectors of data [1]. The main advantage of this estimator over the similar estimators found in the literature is that it can be applied directly on the series of prices as they are observed in the stream without any additional manipulation.

To compute the correlations using the Hayashi-Yoshida estimator in the QualiMaster infrastructure, we use sliding (time) windows of configurable size (e.g., a sliding window of one hour) and advance (e.g., an advance of 10 minutes). For each symbol (stream) and at each tick, we maintain information relevant to the latest interval of the stream, its delta and its overlap with the intervals of every other stream. We calculate the correlation matrix of the input streams whenever a window expire, i.e., when window_size milliseconds have passed. After the correlation matrix calculation takes place, we shift the sliding window by advance seconds. For every stream interval that has now expired —that is, the interval's end is earlier than the window beginning— we cancel its contribution to the estimator. The computation of the correlation matrix is parallelized among the machines available in the QualiMaster infrastructure.

Please note that a detailed description for the specific estimator as well as its implementation within the QualiMaster infrastructure and related examples are described in deliverable D2.1.

2.1.2 Mutual information

Mutual information accounts for non-linear relationships between two data vectors with asymmetrical dependencies. The goal is being able to work on heavy tailed distributions, since this might be the formation of the financial data given to the QualiMaster infrastructure. Handling such higher order statistics [2, 3, 4] imposes a computational complexity, which for mutual information is actually quadratic.

Mutual information is frequently used for retrieving the complete dependence structure between two time series (including nonlinear dependence). It basically computes the data shared between the given time series, which (intuitively) can be seen as measuring how much knowing variable X (or variable Y) reduces uncertainty about variable Y (or variable X).

Given two time series X=$\{\mathrm{x}_1,\ \mathrm{x}_2, \ldots,\ \mathrm{x}_\mathrm{n}\ \}$ and Y=$\{\mathrm{y}_1,\ \mathrm{y}_2\ , \ldots,\ \mathrm{y}_\mathrm{n}\ \}$, then the mutual information of X and Y is computed by the following formula [5]:

$$I(X;Y) = \int_Y \int_X p_{XY}\left(x,y\right) ln \frac{p_{XY}\left(x,y\right)}{p_X\left(x\right) p_Y\left(y\right)} d_x d_y$$

With symbol $p_{XY}(x,y)$ we denote the joint probability density function between X and Y, and symbols $p_X(x)$ and $p_Y(y)$ denote the marginal probability density functions.

The main challenge for evaluating the mutual information formula is on computing the probability density function, and more specifically on estimating this probability. To deal with this issue, several methodologies computing such an estimation have been suggested. Some examples (explained in [5]) are the kernel density estimators, the k-nearest neighbors, the Edgeworth approximation of differential entropy [6], and adaptive partitioning of the XY plane.

For incorporating a mutual information family in the QualiMaster infrastructure we will study existing estimators aiming at computing the probability density function, for example the ones mentioned above such as adaptive partitioning and the kernel density estimators. This is work that we have recently started and plan to progress on it during the following months.

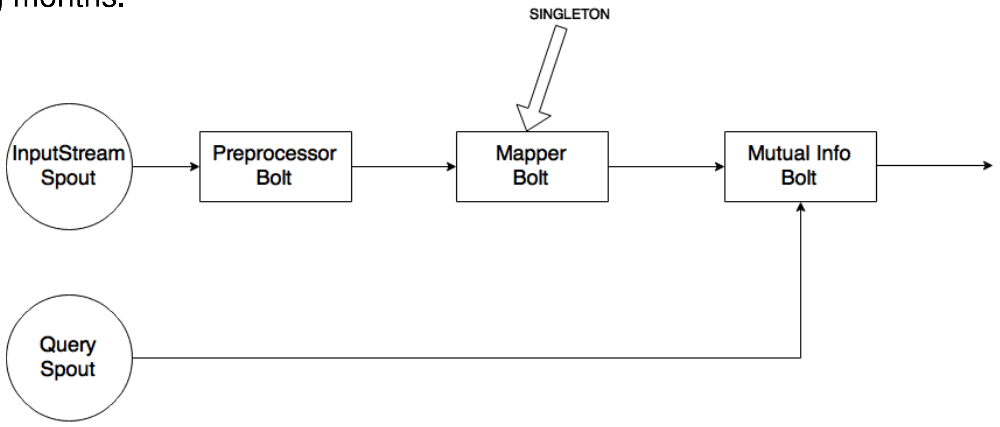


Figure 1: The structure of the Mutual Information storm topology.

Implementation in QualiMaster

The Mutual information algorithm has be parallelized using Storm by distributing the pairs that need to be monitored among the available nodes of the cluster incorporated in the QualiMaster infrastructure. Figure 1 shows the structure of the Mutual Information storm topology. The way this topology works is by firstly receiving the input stream from the spout that is implementing the Spring API to fetch financial data from Spring. Then, the spout forwards the data to a “Preprocessor” bolt that transforms the input data from a single string to a set of specific fields so that the initial tuple can be created (e.g., ⟨MP name, timestamp, value, volume⟩). The formatted tuple is then passed on to a “Mapper” bolt that is responsible for partitioning the pair computation as evenly as possible to the component that follows. Note that “Mapper” bolt needs to be a singleton (no parallelism) and it also needs to receive a “Configuration Stream” during the initialization of the topology in order for it to know the complete list of symbols to be monitored so that it can partition the pairs.

The last component, i.e., the “Mutual Info” bolt, is responsible of performing the main computation of the Mutual Information. Each task of this component will be assigned to compute a subset of all pairs, assigned by the “Mapper” bolt, and tracks the progression of the MI index of all those pairs. Finally, the “Query” spout is responsible for requesting the current MI indexes to be forwarded from the “Mutual Info” bolt to the output of the topology.

The last component, i.e., the "Mutual Info" bolt, is responsible of performing the main computation of the Mutual Information. Each task of this component will be assigned to compute a subset of all pairs, assigned by the "Mapper" bolt, and tracks the progression of the MI index of all those pairs. Finally, the "Query" spout is responsible for requesting the current MI indexes to be forwarded from the "Mutual Info" bolt to the output of the topology.

2.1.3 Transfer entropy: the upcoming estimator in this family

Our plan is to also incorporate transfer entropy [7] as an upcoming approach in the family of the correlation matrix estimators for the QualiMaster infrastructure. Transfer entropy quantifies the amount of information transfer from one variable to another variable. The computational complexity is cubic, becoming a potential bottleneck for a large number of market players. The computational challenge lays on the estimation of the probability density function. To deal with this problem, researchers estimate the probability density function using different methodologies.

We aim at using the infrastructure for efficiently executing transfer entropy, i.e., through possible alternative algorithms for estimating the probability density function are introduced in [8]. In addition, we will define the situations on which this estimator should be invoked instead of the other estimators of the particular family.

2.2 Distributed similarity indexing

2.2.1 Overview of the original StatStream approach

StatStream [9] consists of a real-time, sliding window based framework that aims at efficiently and accurately calculating pairwise correlations above a certain threshold. The basic idea of StatStream relies on, firstly, approximating the original data streams and projecting them into a smaller representation and, secondly, using this smaller representation to index possibly similar streams into neighboring cells of a global index.

The approximation of the original data streams is performed using the Discrete Fourier Transform (DFT). This way, each stream can be represented by a set of DFT coefficients, which can be orders of magnitude smaller than the original stream in terms of size. After the approximation, StatStream uses the first few DFT coefficients from each stream in order to index it to the global index, namely the Grid Structure. The correlation between two streams can be calculated using the Euclidean Distance between the respective normalized streams. Hence, since the DFT transformation preserves the Euclidean Distance, only streams indexed in neighboring cells of the Grid Structure can be considered as possibly correlated (no false negatives). The dimensionality reduction using DFT coefficients along with the pruning provided by the Grid Structure greatly increase the efficiency of the whole system and allow for much larger numbers of streams to be monitored in an online fashion.

The main idea of incorporating this approach in the QualiMaster infrastructure is to partition the load equally among the cluster nodes. So, each node first computes single stream statistics (e.g. moving average, Fourier coefficients, etc.) over the basic windows and then hashes each stream to the grid structure based on its Fourier coefficients. Deliverable D2.1 provided the details for the incorporation of StatStream in the QualiMaster infrastructure.

2.2.2 Enhancements for similarity of financial data

Although StatStream provides a framework capable of computing the pairwise correlation among thousands of time series, it also has certain limitations. StatStream's main limitation lies in the fact that it only relies on the Euclidean distance in order to approximate the correlation. This constrain exists due to the fact that StatStream firstly approximates each time series by a set of Discrete Fourier coefficients. Discrete Fourier coefficients are well known to preserve the Euclidean distance, and using a transformation of the original time series to the normalized time series, the authors of StatStream have managed to link the correlation to the Euclidean distance.

Another limitation of StatStream is the use of a very specific type of index, namely the Grid Structure. Since the Grid Structure uses nearest neighbors to estimate possibly correlated pairs of time series, it also introduces the necessity to use the Euclidean distance as a measure even if the rest of the framework was different and did not rely on it specifically.

T-Storm was inspired by StatStream's limitations. T-Storm's goal is to provide a framework to the end user where the **approximation technique** and the **distance measure** are considered to be a black box, where the user can choose among various different distance measures (e.g., cosine, Hamming, etc.) or even create their own distance measure and incorporate it in T-Storm. In order to do so, the index of the framework is generic and not tied to a specific distance measure (as StatStream's Grid Structure). T-Storm incorporates the use of the LSH index which is a widely used approach for efficient and (tunably) accurate similarity search. The LSH index is distance measure agnostic and can be used independently.

2.3 Querying stock correlation graph

2.3.1 Importance for risk analysis

There were numerous attempts in the research community to define systemic risk [10, 11, 12, 2]. As we discussed in Section 2.1, i.e., "Correlation matrix estimators", one possibility is to construct and maintain a network structure with the data from the available market players that are constructed using, for example, the correlation estimators we have already described. This network is analyzed for potential fallbacks of the market player dependency structure.

For a concrete example of what it means to measure risk analysis, consider the “too big to fail” assessment [13], which is one of the most commonly used measures. The “too big to fail” asserts if certain market players, such as corporations and institutions, are very large and interconnected and thus their failure would also affect a large number of other market players (the players that are directly or closely connected to them). The main objective is to detect such market players. Thus, the economic system (e.g., government) can support them early enough to avoid their failure or to stop the spread of the failure to the related players.

During the last months, we have started working on methodologies that will enrich the QualiMaster infrastructure with the capability to evaluate assessments for measuring systemic risk, such as the “too big to fail” assessment. Our current efforts is on defining and constructing the network structure as well as the basic required methods. This includes a stock correlation graph that can reflect modifications related to the market players as well as efficient execution of graph methods on such evolving graphs. The following paragraphs provide more details on our efforts and current results.

2.3.2 Modeling and possible queries

To formally model volatile graphs, we assume the existence of an infinite set of identifiers V, attributes A, and atomic values D. Each instance provides a value d_i for each attribute a_i of the corresponding relation. The value d_i can be either atomic value, time information, or identifier, i.e., $d_i \in D \cup Z \cup V$.

We consider a relation R that contains information about market players, such as name, stock price, etc. An instance r is a tuple $\langle n, t, d_1, ..., d_k \rangle$, where $n \in V$ corresponds to the instance's identifier, $d_i \in D$ to the attribute values for the particular attribute $a_i \in A$, and $t \in Z$ the time these values appeared for describing the instance (i.e., market player) with identifier n.

Given that the infrastructure deals with streaming data, R contains various instances with the same identifier but with a different time, i.e., r_i, r_j, ..., r_n with $r_i.n = r_j.n = ... = r_n.n$ and $r_i.t < r_j.t < ... < r_n.t$. Each relation r_i appears at $r_i.t$ and is valid until another relation with the same identifier appears, i.e., until r_j with $r_i.n = r_j.n$ and $r_j.t > r_i.t$.

We now explain how to create a graph using the computed correlations among the values sent to the infrastructure by the monitored stock markets. We assume that the identifiers in V correspond to the nodes composing the graphs and that the instances in R provide the information encoded in the nodes. Let V_t denote all the market player information with time equal to t, i.e., $V_t = \{ r_i.n \mid r_i \in R$ and $r_i.t = t \}$. A graph G_t, also called snapshot, denotes the graph occurring at time t using V_t. G_t is given by tuple $\langle t, V_t, E \rangle$, where V_t provides the nodes with their values and $E \subseteq (V_t \times V_t)$ the edges.

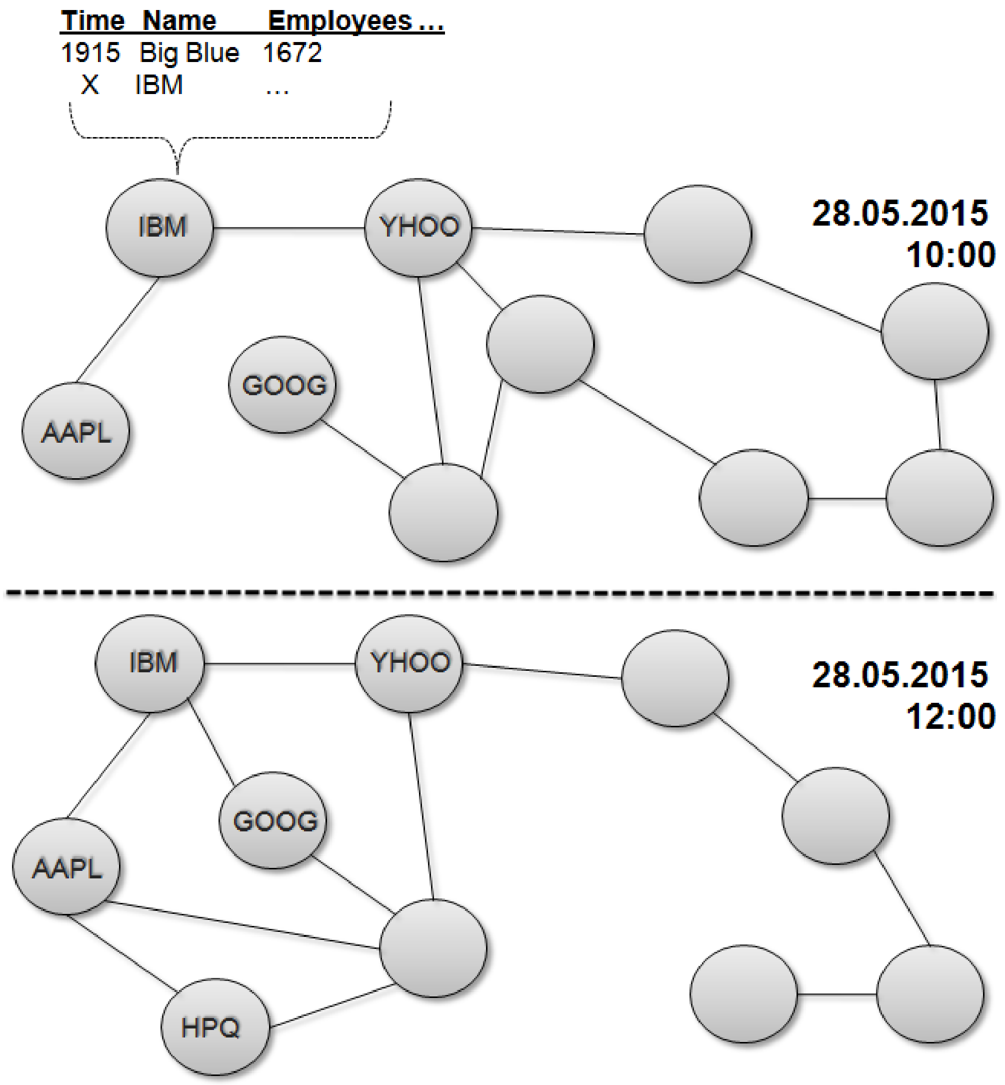


Figure 2: The correlations between market players are used to create a network structure. The nodes, edges, and other information encoded in this network change as time passes, i.e., when the infrastructure receives new values for market players.

Given a particular R, we have a sequence of snapshots, i.e., G_t, $G_{(t+1)}$, ..., $G_{(t+1+i)}$, ... The snapshots illustrate the evolution as time increases, i.e., t, $t+1$, ..., $t+1+i$, ... To achieve our goals, we consider the following modifications between two sequential snapshots:

1. addition of a node, denoted as $\oplus n$;
2. deletion of a node, denoted as $\otimes n$;
3. addition of an edge, denoted as $\oplus(n_i,n_j)$;
4. deletion of an edge, denoted as $\otimes(n_i,n_j)$; and
5. assignment of a new instance for a node, e.g., instance $r_{(i+1)}$ instead of r_i.

An example with two snapshots is shown in Figure 2. Nodes in the graphs are the market players and edges the correlations. As time passes, nodes are assigned new instances, as for example shown with the small table for the node labeled “IBM”. Other modifications shown in the figure are addition and deletion of nodes. Furthermore, the computation of the correlations between the market players imposes additions and deletions of graph edges. For instance, the edge connecting nodes labeled “IMB” and “GOOG” means that the correlation estimator indicated a correlation between the “IMB” and “GOOG” market players at 10 o’clock. The deletion of the particular node in the following graph means that the correlation estimator indicated that there was no correlation between the particular market players at 11 o’clock.

2.3.3 Methodology

Our goal is to be able to execute queries over the snapshots of the evolving graphs, as this is a vital part for assessing systemic risk tasks. Queries can involve: (i) a particular time point that would result in a single snapshot, or (ii) a time period that implies retrieving a collection of snapshots. More specifically, we allow the following grammar for time in the queries:

$$\textsc{Time} \;:=\; t \;\mid\; (t_s, t_e] \;\mid\; [t_s, t_e) \;\mid\; (t_s, t_e) \;\mid\; [t_s, t_e];$$

Query execution considers only the last expression, i.e., [t_s, t_e], since other expressions can be represented using this one. More specifically, t can be represented as [t, t], and expression (t_s, t_e] as [t_s+1, t_e].

Selecting graph elements is primarily based on a direct reference to the particular node (i.e., $n \in V$) or instance (i.e., $r_i \in R$) through their identifiers. Queries can also include conditions on the instances. Thus, the allowed grammar is:

$$\textsc{Criterion} \;:=\; \textsc{node id} \;\mid\; \textsc{instance id} \;\mid\; d_i = string \;\mid\; d_i \text{ like } string;$$

In the introduced query execution algorithms we consider only node ids, and map the remaining expressions into the corresponding nodes with the analogous time periods.

Given our setting with volatile graphs, we have different options with respect to the paths that connect two graph nodes. More specifically, a path query expresses inquires related to paths between nodes *n*α and *n*β that are described by two Criterions. It is defined as:

$$\text{Q}_P\text{:}\quad \langle \textsc{Time}, \textsc{Criterion}_A, \textsc{Criterion}_B, \textsc{ConnectionType} \rangle$$

where ConnectionType denotes the path type. Let P_t = { n_α, ..., n_β } denote a path connecting node n_α with n_β. Then, the path types are defined as follows:

1. **Continuous Path** if there is a path between the two requested nodes in all snapshots of the defined time period. It is defined as: $\forall t \in \textsc{Time} \rightarrow \exists P_t : n_\alpha, n_\beta \in P_t$

2. **Earliest Path** means that the path appears in the snapshot with the earliest appearance. It is defined as: $\exists P_t$ with $n_\alpha, n_\beta \in P_t \wedge t \in \textsc{Time}$: $\forall P_{t'}$ with $n_\alpha, n_\beta \in P_{t'} \wedge t' \in \textsc{Time} \rightarrow t < t'$

3. **Shortest Path** means that the path is the shortest (in terms of involved edges) regardless of the snapshot's appearance time. It is defined as: $\exists P_t$ with $n_\alpha, n_\beta \in P_t \wedge t \in \textsc{Time}$: $\forall P_{t'}$ with $n_\alpha, n_\beta \in P_{t'} \wedge t' \in \textsc{Time} \rightarrow |P_t| \leq |P_{t'}|$

4. **Reachability** requires deciding if such a path appears in at least one of snapshots for the requested period. It is defined as: $\exists\ P_t$ with $n_\alpha,\ n_\beta \in P_t$ and $t \in$ Time.

We are currently working on retrieving these four types of paths in an efficient manner. This primarily involves constructing an indexing structure that encodes the time information and associating nodes with related node versions and neighboring nodes. Path queries will be executed over this indexing structure.

2.4 Summary and Future Work

Our current efforts focus on continuing and finalizing the implementations for the approaches described in the above paragraphs. The finalization of each approach will be followed by an extensive evaluation. These evaluation will be performed with data sets created using the financial data of SPRING as well as synthetic data reflecting specific behaviors. Emphasis will be given not only at quality but also at performance and especially scalability.

In addition to the completion of the approaches and their evaluation, we also plan to investigate other possibilities that might be useful with respect to the goals of the QualiMaster project. For this, we are currently considering count min sketches that can be used for maintaining frequencies and frequency moments, and ECM sketch that provides compact data stream summaries over both time-based and count-based sliding windows with probabilistic accuracy guarantees. [14]. We are also considering incorporating measures for volatility [15], i.e., computing the degree of price movement market players.

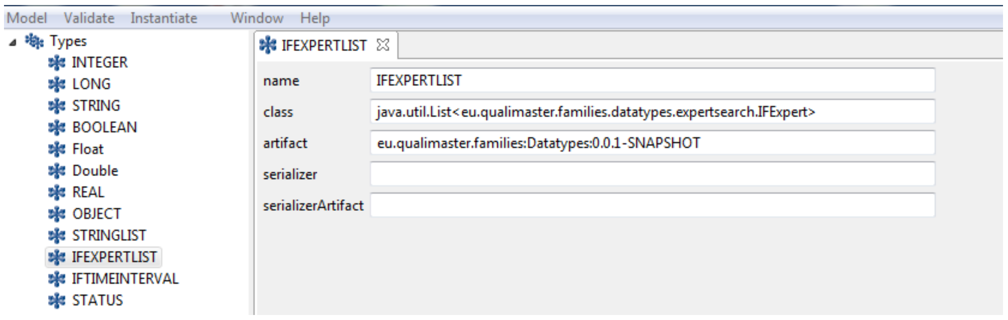
3 Components for (Social) Web Data Analysis

The advances in the computing science and technology of the past decade brought to the human beings the ability to communicate and interact with each other to a degree not even thinkable before. The rise of the social media thus offers a plethora of different points of view on the sociological aspects of the humanity or the particular interests of a sub group of people. It is the latest that we want to explore using appropriate algorithms applied to appropriate sources targeting the financial domain.

The social media can be categorized for example according to the type of the used material, say text centric (Twitter), image centric (Instagram) or in between (Facebook, Tumbler). Another classification can be done based on the life cycle of the posts. While Twitter messages are short and likely focused following tight the events they refer to, blog posts (to give the other extreme) may be long and of more general nature. As we aim to follow the evolution of the financial market and of the individual players in it, the short, focused text messages are of great interest for us. Such messages are perfectly suitable for a stream based solution. Therefore we adopt Twitter as the source for the data extraction in the first place, thought the algorithms that we use are as well applicable to other sources that we plan to include in one of the next steps.

Component Interface

As the interfaces of new components from the (Social) Web Data Analysis considers new specific types supported in D4.2, in this section, we discuss the generated interface for Twitter PreFiltering component as an example. We will not repeatedly describe all component interfaces in other sections, as the basic interface generation process is similar to this example. If needed, please refer to this section.



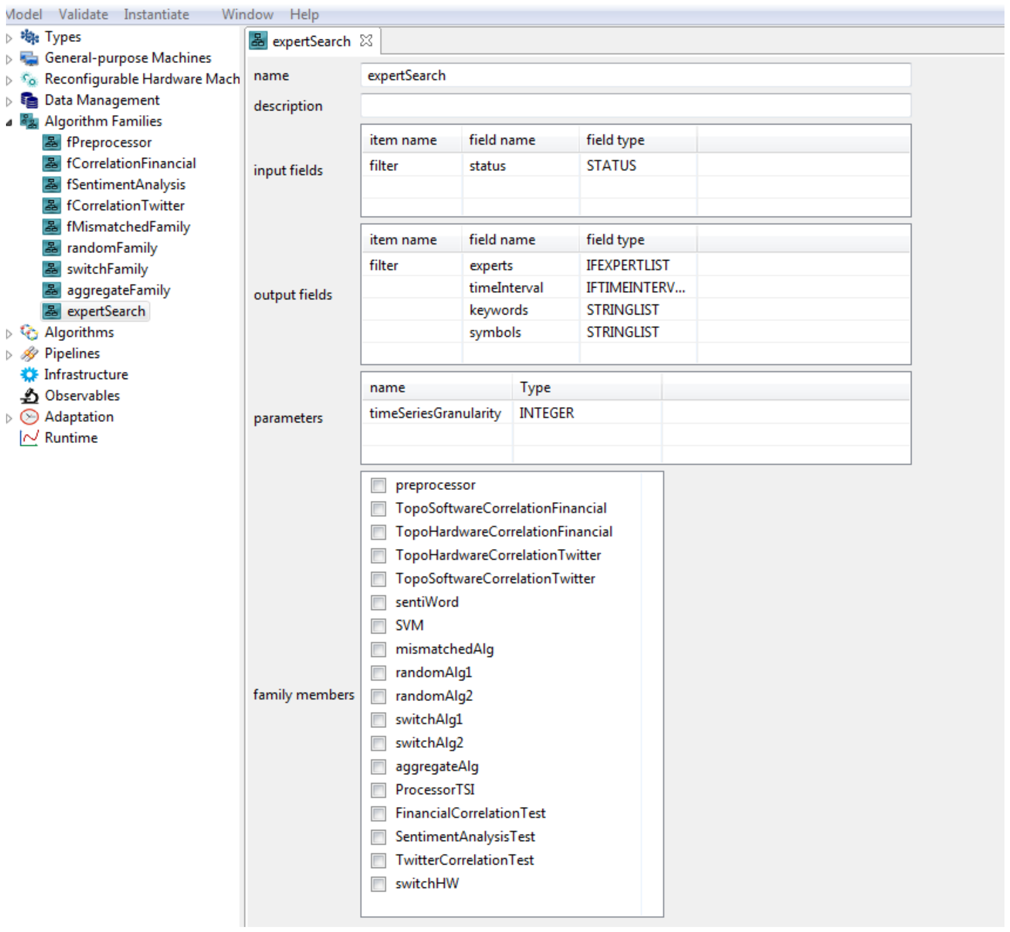
name	IFEXPERTLIST
class	java.util.List<eu.qualimaster.families.datatypes.expertsearch.IFExpert>
artifact	eu.qualimaster.families:Datatypes:0.0.1-SNAPSHOT
serializer	
serializerArtifact	

Figure 3: Types editor for configuring the specific type of the list of IFExpert.

The derivation of component interface with the specific types mainly takes three steps, namely:

- Configuration of the specific types used in the component interface.
- Configuration of the component interface.
- Instantiation of the configured component for deriving the interface Java code.

In the Twitter PreFiltering Component, there are three specific types need to be configured, i.e., the list of IFExpert, IFTimeInterval as well as Status. As an exam-



name: expertSearch

description:

input fields

item name	field name	field type
filter	status	STATUS

output fields

item name	field name	field type
filter	experts	IFEXPERTLIST
	timeInterval	IFTIMEINTERV...
	keywords	STRINGLIST
	symbols	STRINGLIST

parameters

name	Type
timeSeriesGranularity	INTEGER

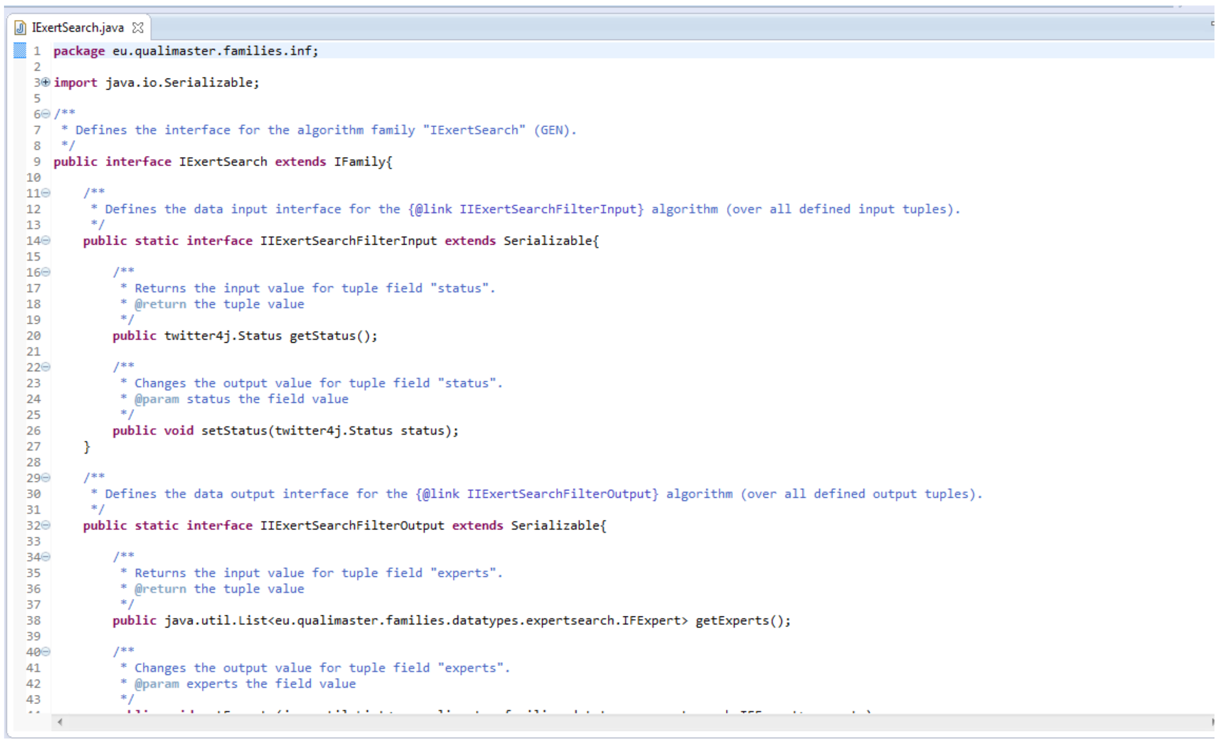
family members

- ☐ preprocessor
- ☐ TopoSoftwareCorrelationFinancial
- ☐ TopoHardwareCorrelationFinancial
- ☐ TopoHardwareCorrelationTwitter
- ☐ TopoSoftwareCorrelationTwitter
- ☐ sentiWord
- ☐ SVM
- ☐ mismatchedAlg
- ☐ randomAlg1
- ☐ randomAlg2
- ☐ switchAlg1
- ☐ switchAlg2
- ☐ aggregateAlg
- ☐ ProcessorTSI
- ☐ FinancialCorrelationTest
- ☐ SentimentAnalysisTest
- ☐ TwitterCorrelationTest
- ☐ switchHW

Figure 4: Families editor for configuring the interface of the Twitter PreFiltering component.

ple, Figure 3 illustrates the configuration editor of the IFExpert list type in the QualiMaster Configuration Tool. The configurable elements shown in the Types editor can be referred to the discussion of the arbitrary field types in D4.2. As depicted in Figure 3, we configure a descriptive name IFEXPERTLIST for the specific type of the IFExpert list. Further, we specify a fully qualified class name $java.util.List < eu.qualimaster.families.datatypes.expertsearch.IFExpert >$ indicating the implementation class name of List in the Java library as well as the implemented IFExpert type. Moreover, the artifact eu.qualimaster.families:Datatypes: 0.0.1-SNAPSHOT deriving the Maven build specifications for IFExpert type needs to be configured as well.

The configuration of the component interface actually relates to the configuration of the algorithm family discussed in D4.1. Basically, the algorithm families group data processing algorithms with similar functionality, but different quality characteristics and tradeoffs. In this deliverable, individual components can be reflected to the algorithms grouped to the certain family. As illustrated in Figure 4, the so-called expertSearch family is considered as the family of the Twitter PreFiltering Component, which is configured by its input/output item types as well as the functional parameter. As the interface needs to be derived before the implementation of the family member, thus, here the family does not contain



IExertSearch.java

```java
package eu.qualimaster.families.inf;

import java.io.Serializable;

/**
 * Defines the interface for the algorithm family "IExertSearch" (GEN).
 */
public interface IExertSearch extends IFamily{

    /**
     * Defines the data input interface for the {@link IIExertSearchFilterInput} algorithm (over all defined input tuples).
     */
    public static interface IIExertSearchFilterInput extends Serializable{

        /**
         * Returns the input value for tuple field "status".
         * @return the tuple value
         */
        public twitter4j.Status getStatus();

        /**
         * Changes the output value for tuple field "status".
         * @param status the field value
         */
        public void setStatus(twitter4j.Status status);
    }

    /**
     * Defines the data output interface for the {@link IIExertSearchFilterOutput} algorithm (over all defined output tuples).
     */
    public static interface IIExertSearchFilterOutput extends Serializable{

        /**
         * Returns the input value for tuple field "experts".
         * @return the tuple value
         */
        public java.util.List<eu.qualimaster.families.datatypes.expertsearch.IFExpert> getExperts();

        /**
         * Changes the output value for tuple field "experts".
         * @param experts the field value
         */
```

Figure 5: Interface fragment of the Twitter PreFiltering Component.

any member. After all needed information configured, the interface Java code depicted in Figure 5 can be generated by the instantiation process explained in D5.2. Next we will discuss the implementation details for each component.

3.1 Twitter PreFiltering Component

We now describe our progress on a component focusing on pre-processing and filtering Twitter data streams. In particular, we have focused on developing a Java MapReduce module for filtering out human and machine non understandable content, also referenced as “low-quality” content in the literature [16, 17, 18]. This is an initial approach for the component and it will be extended during the course of the project. The remainder of this section is organized as follows. We first provide a quick introduction regarding the need of a stand-alone component for pre filtering the Twitter data streams. Next, we present the steps followed in our approach, as well as the diagram showing the functionality of the component. Finally, we perform an evaluation task in order to assess the running time of the software module.

3.1.1 Introduction

In the last years Twitter has become one of the most popular social web platform for spreading information covering various events, braking news, or simply sharing personal thoughts online. While this implies a lot of real-time useful information coming out online, a non-negligible amount of the tweets share content which is usually not the target of the algorithms mining the social streams [19, 20]. In order to ease and structure the access of the machine learning algorithms to useful content, the work in this section has focused on developing a Java MapReduce module for filtering out human and machine non understandable content. Our methods here focus on filtering out the so called “low-quality” content. A review of previous literature reveals various works using preliminary Twitter filtering steps for a wide range of different problem settings [16, 17, 18]. Within this work we focus on a component aimed at pre-filtering the Twitter data that is used by the sentiment and event detection algorithms.

3.1.2 Approach

The overall approach which is implemented in Java MapReduce consists of 6 steps which are described below, as well as visually depicted in Figure 6.

Step 1: Remove tweets starting with ‘RT’ (retweets).
Step 2.1: Discard the words containing a reference (starting with character “@”), links (starting with “http” or “https”), hashtags and emoticons.
Step 2.2: Keep only the words that have at least 3 characters but less than 15 (terms being too short or two long are claimed to carry less meaningful information [17, 20]).
Step 2.3: Stemm the textual content. Keep the messages that after applying Step 2, still contain at least 3 words.
Step 3: Remove duplicate tweets, a common source of spam content.

Step 4 (optional): Aggregate tweets over a predefined input information criterion *IC* (e.g., hashtags). Compute the Jaccard similarity score of the tweets within each IC Group. Remove the groups of tweets for which the similarity is above the given threshold *k*.

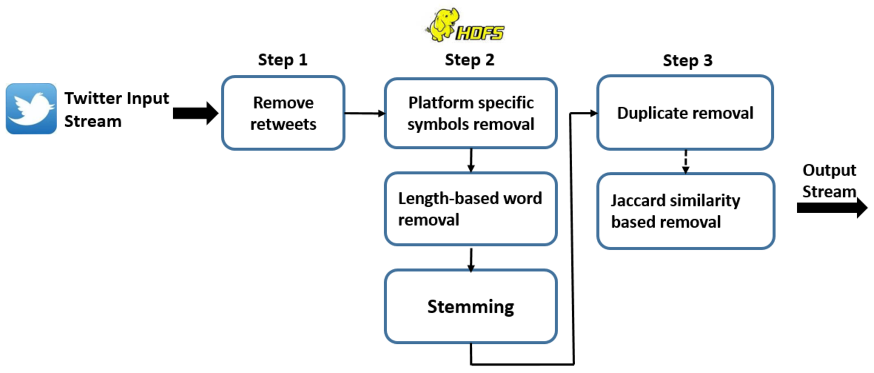


Figure 6: Twitter Stream PreFiltering Component.

Table 2: Running Time vs. Number of Remaining Tweets for the SentiHashtags Dataset.

No. Tweets	CPU Time (seconds)	No. Remaining Tweets
		262,209,417 (all English tweets)
Step 1	613	169,666,209
Step 2.1	617	168,649,262
Step 2.2	618	167,897,137
Step 2.3	618	133,603,633
Step 3	621	133,496,598
Step 4	1,079	16,512,151

3.1.3 Evaluation

Time Behavior. Time behavior, as reported in D2.1 Section 2.4 represents the time-related sub-dimension of performance efficiency. In this subsection we focus on the latency aspects of the MapReduce component. In Table 2, we provide statistics about the running time vs. the number of remaining tweets for the MapReduce Twitter PreFiltering component applied on the 6 months Twitter Stream API dataset that is described in Section 3.2. We observe that it requires about 10 minutes to perform the Steps 1 to 3 on a set of 262 million input instances, while the Step 4 itself takes 7.63 minutes on the remaining set of 133 million tweets. The MapReduce Software component was run on a Hadoop Cluster having 1.25TB Total Memory and 244 VCores.

3.1.4 Summary

Our work here aimed at building a MapReduce software component able to systematically remove tweets that rarely have useful information. To this end, most of the removal steps are based on approaches coming from related work. The library itself is being used by the EventDetection and Sentiment Analysis components. Directions of our future work involve implementing the component using Big Data real-time stream processing approaches.

3.2 Mining, Analyzing and Detecting Hashtags carrying Sentiment Information

We now describe our progress on analyzing and exploiting Twitter content that carries opinionated information. In this work we implemented a first approach for identifying hashtags matching opinionated content. Finally, we performed a first user annotation study in order to evaluate the effectiveness of the method for identifying opinionated hashtags. The goal of this work is to improve the methods that aim at harvesting opinionated content, as well as automatically generating training instances for the machine learning approaches dealing with the sentiment classification tasks.

3.2.1 Introduction

Sentiment Analysis is the area which deals with the problem of assigning opinion labels (e.g., “positive” vs. “negative” vs “neutral”) to various types of documents using diverse text-oriented and linguistic features. Nowadays, Twitter represents a major platform accessible to almost everyone and everywhere, for sharing opinions about all sort of topics and events. While a lot of work has focused on tweet-based sentiment assignment (a comprehensive state-of-the-art available in D2.1, Section 6), different models have proved their limitation mostly due to the ambiguity, shortage in information that a 140 characters text can deliver and the platform specific language. In the same time, most of the application scenarios nowadays recall the need of per entity sentiment aggregation. Despite the large number of studies that focused on sentiment classification on various types of text, the potential of opinionated hashtags associated with the shared content still remains unexplored in the context of sentiment classification. In this section, we first study the presence of hashtags within a large dataset containing 1% out of all Twitter messages posted in the World during a six months period. Next, we study a first approach for identifying hashtags carrying opinionated content. Finally, we perform a first user annotation study in order to evaluate the effectiveness of the method for identifying opinionated hashtags.

3.2.2 Related Work

Davidov et al. [16] analyze the performance of classification models for predicting the “sentiment types” of Twitter messages, where the sentiment types are defined as a fix set of hashtags and smileys. The experimental setup uses 50 Twitter hashtags, each hashtag containing a word suggesting an opinion (e.g., *#happy*, *#crazy*, *#bored*) and 15 mood indicating emoticons as sentiment labels. The final goal here was to predict (assign) previously unseen tweets to one of the hashtags or emoticons class. In a recent work [18], the authors employ various methods in order to predict the hashtags sentiment polarity on a set of hashtags containing sentiment words (e.g., *#iloveipad*, *#ihateipad*). In contrast, our work aims at identifying hashtags carrying sentiment information independent from the actual textual content of the hashtag itself. Table3 shows two example hashtags as identified by our component for being positively (*#surface3*) respectively negatively opinionated (*#upssucks*).

Table 3: Examples of a positive (left) and a negative (right) hashtag as identified by the machine learning approach based on the existing tweets.

#surface3	#upssucks
Microsoft reveals a thinner, faster surface pro 3 tablet (video) http://t.co/gmjbrenusr #itbnews #surface3 #microsoft	shout out to ups for losing one of my boxes and shipping the rest to the wrong address #movingpains #upssucks #happymonday
#beautiful new pink and black #microsoftsurface case by cornercovers http:t.cozotjxc7xrh via etsy #etsy #handmade #surface3	been a year. forgot the torture of #ups in queens. no doorman=no package. customer service is awful. never again. #upssucks #useless
finally got it right - http:t.colwk7kpnmtg #surface3 #surfacepro3	idiots at ups they come up my driveway and don't deliver my package. i understand thursday but 4 days now #upssucks #idiots

3.2.3 Dataset

Data Gathering. We used the Twitter Stream API which provides a random sample comprising of 1% of all tweets in the world per daily basis. In order to form our collection, we collected all messages together with the available meta-information published in JSON format via the Stream API between January and June 2014. This process yielded a collection of 784 million tweets from which 262 million are in English.

Data Characteristics. In Table 4 we provide a couple of preliminary statistics regarding the collection size with respect to the number of tweets, hashtags and users per month. We can observe that the number of tweets per month ranges from 122 to 134 million. Table 4 also shows that our sample comprises of about 3 million hashtags per month, and approximately 20 million tweets per month contain one or more hashtags. Overall, 115 million tweets contain a hashtag out of which 45 million are in English.

For a more detailed inspection, Figure 7 shows the distribution of hashtags for the set of 115 million tweets containing a hashtag, with an average of 1.12 hashtags per tweet. The following observations can be made. First, over 65% of the tweets contain one hashtag only. The messages containing more than one hashtag but less than five represent about 30% of the population for the 115 million sample, indicating that a non-negligible fraction of messages can be exploited within a tag co-occurrence environment. Finally, tweets with more than five hashtags represent about 4% of the population. A closer inspection of this sample revealed that most of the tweets from this set tend to be automatic machine generated content.

Table 4: Statistics for the Twitter Stream Dataset.

Month	Total No. of tweets	No. Hashtags	No. Distinct Hashtags	No. tweets with Hashtags
January	129,439,833	27,141,523	2,751,917	16,345,146
February	122,695,962	29,339,317	2,734,703	17,102,059
March	131,155,583	36,565,267	3,082,665	20,475,874
April	133,297,542	37,426,142	3,034,886	20,633,781
May	130,382,142	37,088,549	2,934,287	20,247,694
June	134,470,862	37,775,114	2,895,408	20,699,382
Total	781,441,924	205,335,912	17,433,866	115,503,936

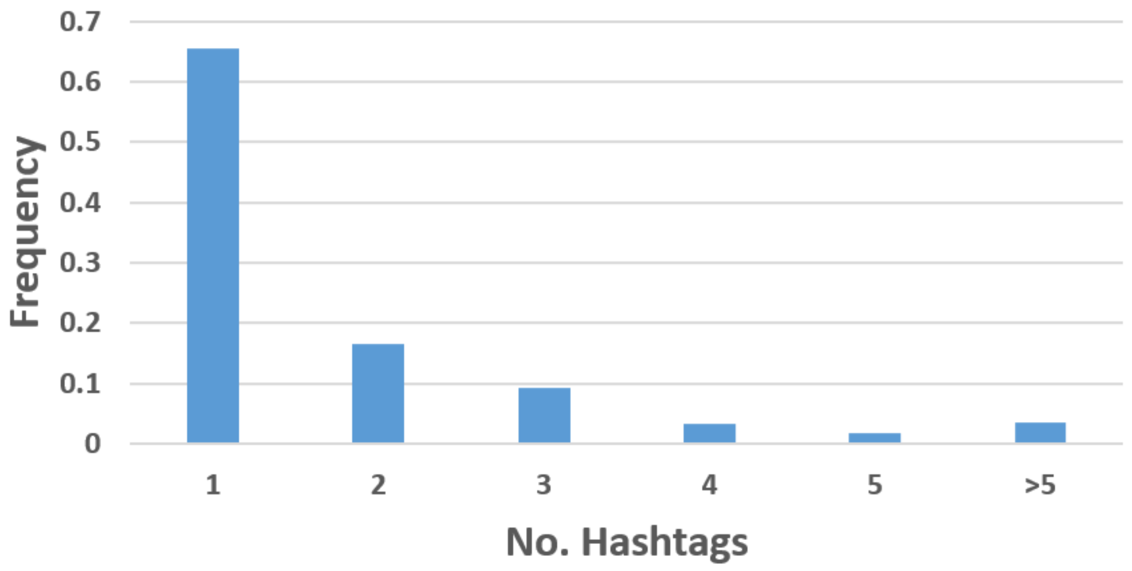


Figure 7: Distribution of hashtags for the 110 million tweets.

3.2.4 Approach

Detecting Opinionated Hashtags. For the Opinionated Hashtag discovery scenario, we first built three types of binary classifiers to separate each sentiment class from the other two classes, i.e. we applied a "one vs. all" (OVA) strategy: positive vs. all, negative vs. all and objective vs. all. The training dataset used was the general purpose sentiment dataset containing 12,000 tweets, with 4,000 tweets in each sentiment class, which was previously described in D2.1, Section 6.1.4. The feature vectors were constructed using the tf-idf weights of the terms appearing in each twitter message. While doing this, we also accounted for negations (i.e., if a negation, say, "not", immediately precedes another term *t*, we created a virtual term not_t).We used balanced sets with equal number of selected instances from each class. For instance, as 4,000 tweets are annotated as positive, the positive vs. all classifier was trained with 4,000 tweets from the positive class and 2,000 tweets selected from each of the negative and objective classes.

Next, we applied the one-vs-all trained sentiment models in order to predict the sentiment class of a hashtag. Within this first setup, we considered all hashtags appearing in at least five tweets which passed through the filtering stages performed by the Twitter PreFiltering component. In a nutshell, this meant focusing on predicting the sentiment of hashtags that appear in non "low quality" tweets that are in English and were not automatically generated by diverse applications connected online. For each hashtag h appearing in at least five tweets, we predicted the sentiment class based on the aggregated scores of the tweets containing the hashtag, as provided by the sentiment classifiers. Next, we ranked all the hashtags for each class based on the aggregated distance from the separating SVM hyperplane of the tweets containing the hashtag. Algorithm1 shows the steps performed by the component in order to filter and compute the positivity scores of the hashtags in the input collection.

3.2.5 Evaluation

For a preliminary systematic evaluation of our strategy for opinionated hashtag discovery, we selected the top-50 hashtags as identified by the classifiers for the classes positive, negative and objective and conducted a user study. The annotator was given a shuffled list of the resulting 150 hashtags, together with the available tweets and asked to label the hashtags as being positive, negative or objective. To this end, we employed two in-house annotators (two computer science researchers). 88% (133 hashtags) respectively 86% (130 hashtags) of the labels assigned by the annotators overlap with those assigned by the machine learning algorithm. Furthermore, we computed the Fleiss' kappa coefficient for the inter-user agreement, which is a measure for computing the agreement between annotators, given a set of category ratings for a fix number of classified instances [21]. The Fleiss's Kappa inter-user agreement among the two annotators was found to be 0.81. Note that according to Fleiss definition, k <0 corresponds to no agreement, $k = 0$ to agreement by chance, and $0 < k \leq 1$ to agreement beyond chance.

Algorithm 1: Detecting Opinionated Hashtags.

```
Input: Large Collection of tweets;
Output: Hashtags - Opinion Score Pairs;
begin
    PreFiltering;
    $tweets \leftarrow input$;
    $filteredtweets \leftarrow []$;
    while $tweet \in tweets$ do
        $text \leftarrow tweet[text]$;
        $lang \leftarrow tweet[lang]$;
        if $text[0-1] \neq rt$ and $lang = en$ then
            $tokens \leftarrow Tokenize(text)$;
            $filteredText \leftarrow []$;
            while $token \in tokens$ do
                if $token[0] \neq @$ and $token[0] \neq \#$ and $token[0-4] \neq http$ then
                    if $Length(token) \geq 3$ and $Length(token) < 15$ then
                        $Push(filteredText, token)$;
            if $Length(filteredText) \geq 3$ then
                $Push(filteredtweets, tweet)$;
    $tweetsBuckets \leftarrow GrouptweetsBasedOnHashtag(filteredtweets)$;
    while $tweetsBucket \in tweetsBuckets$ do
        $tweetsBucket \leftarrow SorttweetsBasedOnTweetText(tweetsBucket)$;
        while $tweet \in tweetsBucket$ do
            $nextTweet \leftarrow NextTweet(tweetsBucket)$;
            $similarity \leftarrow Similarity(tweet, nextTweet)$;
            if $similarity \geq 4.0$ then
                $Remove(tweetsBucket, nextTweet)$;
            else if $similarity \leq 1.0$ then
                $break$;
    Classification step;
    $tweets \leftarrow Flatten(tweetsBuckets)$;
    while $tweet \in tweets$ do
        $text \leftarrow tweet[text]$;
        $text \leftarrow Stemm(text)$;
        $score \leftarrow PosVsAllClassifier(text)$;
    Opinionated hashtags;
    $tweetsBuckets \leftarrow GrouptweetsBasedOnHashtag(tweets)$;
    $opinionatedHashtags \leftarrow []$;
    while $tweetsBucket \in tweetsBuckets$ do
        $avgScore \leftarrow Avg(tweetsBucket)$;
        $Push(opinionatedHashtags, Key(tweetsBucket), avgScore)$;
```

3.2.6 Summary and Future Work

In this section we presented a first approach implemented in Java MapReduce aiming at detecting hashtags carrying opinionated content within large datasets.

We have several future work directions for the exploration and exploitation of opinionated hashtags in Twitter datasets. First, we will aim at developing more complex models for predicting the sentiment class of a hashtag. Second, we will adapt our machine learning methods to the specific requirements that appear within financial domains datasets. Directions of our future work also involve investigating the benefits of hashtag sentiment detection in a couple of scenarios such as result aggregation and trend analysis for the financial domain. We will also focus on studying the temporal development of opinions towards financial entities based on the hashtags carrying sentiment information.

3.3 User and Social Network Analysis

We now describe in detail the methods used for analyzing the content users posted within the monitored social networks. In this deliverable we will focus on the calculation of user influence based on meta information available for each user and the modelling of the user topics describing the domains where users are more active contributors.

3.3.1 Introduction

The modelling of user expertise and influence is important in our scenario because of the availability of large amounts of data which are not of equal relevance for the given domain of stock markets and systemic risk. Dependent on the way a certain user uses the social network we may want to take the contributions into account or not.

Figure 8 shows that a lot of content shared on the Web is not of big interest in our scenario, the main topics with relevance for Systemic Risk and the Stock market (Politics and Finance) are only 22% or the shared messages. Additionally many users have different purposes in mind when using social media[1].

User Models

Besides the stored tweets and the corresponding annotations, we generate an aggregated profile as described in D2.1. In D2.1 we described a strategy for generating this profile based on the top categories of Wikipedia. Additionally we introduce here a profile based on the Thomas Reuters Business Classification (TRBC)[2]. This System contains a

[1] http://www.addthis.com/press/addthis-q3-analysis-reveals-steady-increase-in-mobile-content-engagement#press-release

[2] http://thomsonreuters.com/en/products-services/financial/market-indices/business-classification.html

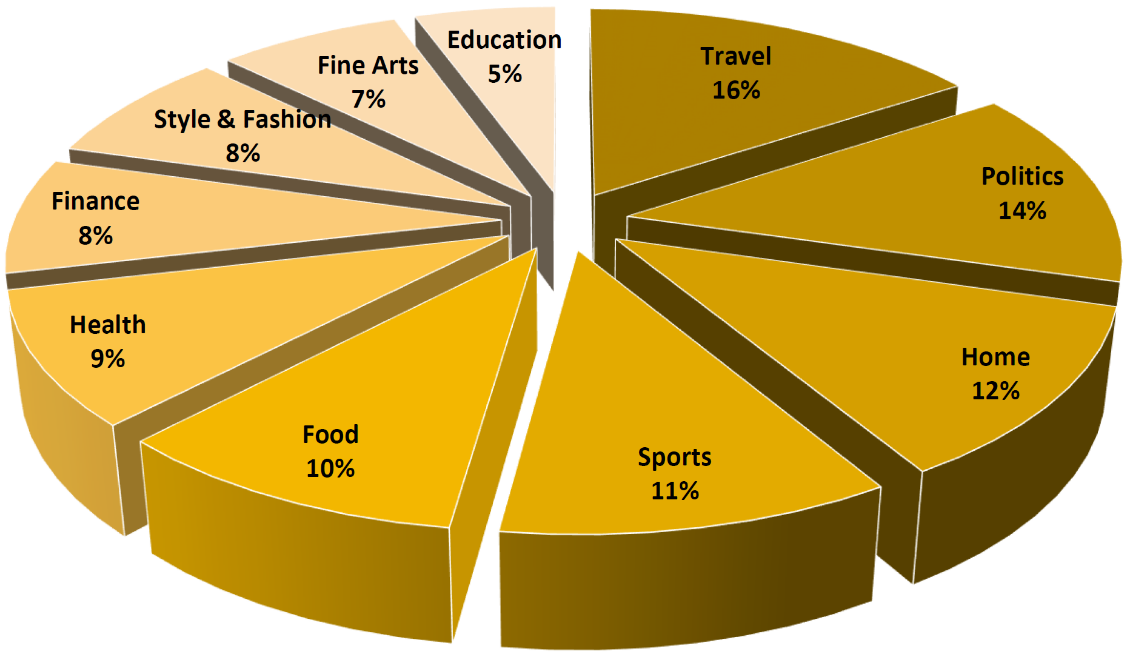


Figure 8: Top 10 shared content on the web

hierarchy for different business sectors, allowing us to model the interests and expertise of the monitored users in a way, which is closer to our areas of interest. We modelled the Economic sectors, Business sectors and Industries to corresponding Wikipedia categories. These different categories model various aspects of the stock market in different granularities. For instance, the Economic Sector Basic Material is split into Business Sectors like Chemicals and Mineral Resources. These are further groups into Industries such as Agricultural Chemicals or Steel. Our aggregated user profiles together with the different levels of granularity allows us to find and monitor users for various scenarios.

3.3.2 User Expertise Analysis

For the analysis of the user expertise we presented a relative general model to monitor the expertise of users based on LDA and entities in D2.1. The entity driven model was further developed to match the special needs of the QualiMaster project. Due to focus of the project on markets and systemic risk we build a user's model focusing upon the Thomson Reuters Business Classification System. This approach allows us to model the expertise and interests of users directly in the domains we are interested in.

We now describe how the creation of user's profiles for defined topics is done. This step consists of 4 major steps.

- Topic selection
- Messages enrichment
- Entity linking
- User finalization

Every generated profile indicates which market sector a user talks about most.

Topic selection The Thomson Reuters Business Classification System builds the base for the topics of the generated profiles, it contains a hierarchy for different business sectors, allowing us to model the interests and expertise of the monitored users in a way that indicates their relatedness to companies of the different stock market sectors. We modelled the Economic sectors, Business sectors and Industries of the TRBC to corresponding Wikipedia categories. Based on these Wikipedia categories the user's interests are described. As a result, our profile only indicates the relatedness of the user to different sectors of the stock market. These user profiles together with the different levels of granularity allows us to find and monitor users for various scenarios. For instance when monitoring companies telecommunication companies we could select users with profiles related to technology.

Messages enrichment In this step we annotate all tweets of the user using the Wikipedia Miner Toolkit [22]. The detected entities are used for the profile creation. The tool provides us links to Wikipedia articles. The links discovered by Wikipedia Miner have a similar style to the links which can be found inside a Wikipedia article. Not all words that have a related article in Wikipedia are used a links, but only word which are relevant for the whole topic are used as links. Since tweets are rather short it is relative hard to define the context, to deal with this problem we can consider larger sets of tweets from one user.

Entity linking The third stage relates the entities that have been mentioned in the users tweets to the beforehand chosen categories representing the different sectors of the stock market. For each of the entities we calculate the relatedness to every article belonging to the beforehand chosen categories. As relatedness measure we use the one proposed by Milne et al. [23], this measures is modelled after the normalized Google distance measure [24] and calculates the ratio between the inlinks two articles have in common and the overall articles linking to them. In order to reduce the influence of articles being mentioned several times we used the log of the number of articles as weight for the profiles. The formula for the relatedness is defined as:

$$R(a,b) = \frac{\log(max(|A|,|B|)) - \log(|AB|)}{\log(|W|) - \log(min(|A|,|B|))}$$

Here a and b are the two articles, A and B are the sets of all articles that link to a and b respectively, and W is the set of all Wikipedia articles. The intuition behind this formula is that articles linking to booth a and b, indicate some relation between a and b while articles with only one or the other suggest the opposite.

Algorithm 2: Creation of User Profiles

```
Input: $T$: Set of tweets of a User $C$: Set of Categories for the Profile
Output: User Profile $P = (T, C)$
1  begin
2      foreach $c_k \in C$ do
3          $ILC \leftarrow (c_k, getInlinks(c_k))$
4      foreach $T_i \in T$ do
5          $A_i \leftarrow getAnnotations((T_i)$
6          foreach $a_j \in A_i$ do
7              $ILA_i \leftarrow getInlinks((a_j)$
8              foreach $ilak \in ILA_i$ do
9                  foreach $ilc_k \in ILC$ do
10                     $R \leftarrow calulateRelatedness(ilc_k, a_j)$ $updateProfile(ilc_k, R)$
11     return $(V, E)$
```

User finalization: In the final stage, we perform a linear aggregation over all of the tweets of a user in order to generate the final user profile. The generated user profile displays the topics a user talks about and together with features like the number of followers or how focused a user is in a certain topic we also get an idea about the expertise of the user in this domain.

Algorithm 2 depicts the steps in the creation of a user profile. The several loops in the algorithm require some time for calculation, to speed up the process our implementation contains a buffer storing the weights for all categories for every entity. This reduces the three inner loops to a single lookup in a hashmap, making the method applicable also for very large sets of users and tweets. Hence, our approach presented here is scalable.

3.3.3 Dataset

The dataset gathered for our experiments consists of around 5,3 Million tweets related to more than 3000 Stocks. These tweets were collected between October 2014 and May 2015 using the Twitter Streaming API together with a set of filters. The filters were selected based on the stock symbols of the different companies (For instance $AAPL for Apple). The idea behind this approach is, to collect a series of tweets which contain a direct relation to a stock market company, and are due to that, of high interest for our domain. Our tweets were written by 333704 different users which overall posted more than 2.4 Billion tweets. Table 5 lists the details of the used datasets.

Out of this large amount of users we randomly selected 10000 for further analysis. For each of the users we downloaded up to 2400 of the last tweets, resulting in a dataset of 12,308,376 tweets.

Table 5: Statistics about Stock related dataset.

	Items	Annotations ALL	Annotations Relatedness >0.5
tweets	5,305,029	30,000,000	4,488,789

Table 6: Statistics Users in Stock related dataset.

	Items	Avg. tweets per User	Avg. Friends per User	Avg. Follower per User	Avg Listed per User
Users	333,704	7389.98	918.57	2138.47	29.59

3.3.4 Efficiency

The efficiency of the proposed methods is based on different selectable features. Most important is the method used for annotating the collected tweets. Here we compared in D2.1 Wikipedia Miner and Illinois Wikifier and showed that Wikipedia Miner perfermed good in terms of processing time. Another important feature is the context size. In our scenario, we are dealing with single tweets but in order to increase the annotation quality the use of more than just one tweet is possible. Our experiments showed that this is not required since the quality based on single tweets is already relative high. Additionally Wikipedia Miner allows the selection of annotations based on the estimated quality of the annotations based on the relatedness between the selected entities. Selecting annotation based on their relatedness allows us to vary the number of annotation that needs to be processed. Figure 9 shows how the number of detected entities per tweet varies with the relatedness.

We can see that the quality is relative homogeneous distributed over the detected entities; this allows us to vary the number of entities we are taking into further account relative free.

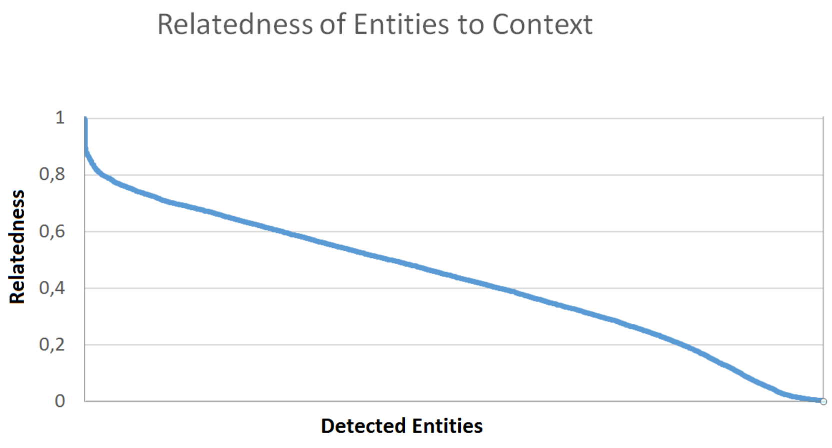


Figure 9: Relatedness of Entities within tweets

Quality

In order to evaluate quality, we first generated profiles for users and companies. These profiles are compared based on different metrics like cosine similarity or various correlation metrics. Once the most similar users for a certain company are defined, we evaluate the quality based on the number of relevant tweets. If a tweet is relevant or not can be measured using different methods. One method is to verify if the user talks about the company of interest by checking for occurrences of the stock symbol. Additionally we manually checked various tweets and estimated the relevance. Another method for automatic relevance calculation is using IR methods as if TF/IDF based measures. In addition, the use of background knowledge is possible like the building of word sets related to a company by using Wikipedia or Dbpedia. Especially Dbpedia offers us to generate sets of words representing the products or services a company offers. The current results of the evaluations performed in terms of quality are described in the following paragraphs:

Topics and Expertise

Figure 10 shows three example profiles for a user which is focused in one area, a user with a focus on some topics and a user that is not focused in any special topic in comparison. A user profile like the one displayed for the *focused user* shows strong relation to technology related topics and / or companies and is focused on this topic, so most of the posted messages are only about one topic. In comparison to this User - the medium *focused user* shows some interest in the area of finance, banking and general, but is not as much concentrated on this topics as the *focused user*. The *unfocused user* shows no clear focus towards any of the topics modelled by the profile.

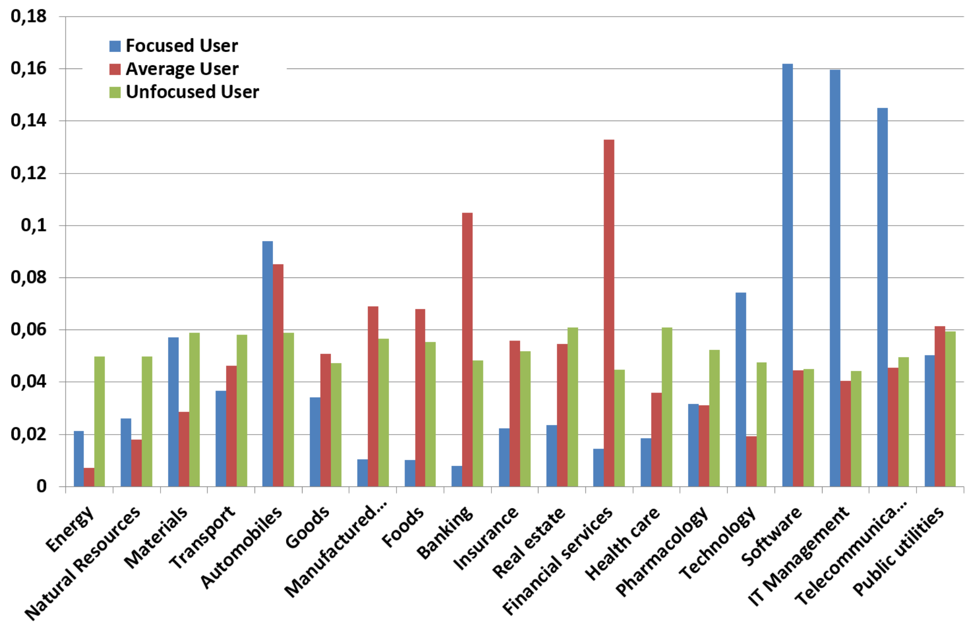


Figure 10: Examples of 3 different users and the corresponding profiles.

We conducted a series of experiments for analyzing whether the users tend to write about many topics or tend to focus on a few topics. Figure 11 shows the distribution of 19 different topics based on business sectors. The users are ordered based on the standard deviation of their profiles thus more *focused users* are on the left side of the diagram. We can see a correlation between the topics users talk about and how focused their profiles are. For example users writing about Health Care and Pharmacology seems to be very focused on this topic while users writing a Public utility seems to be not focused on only this topic.

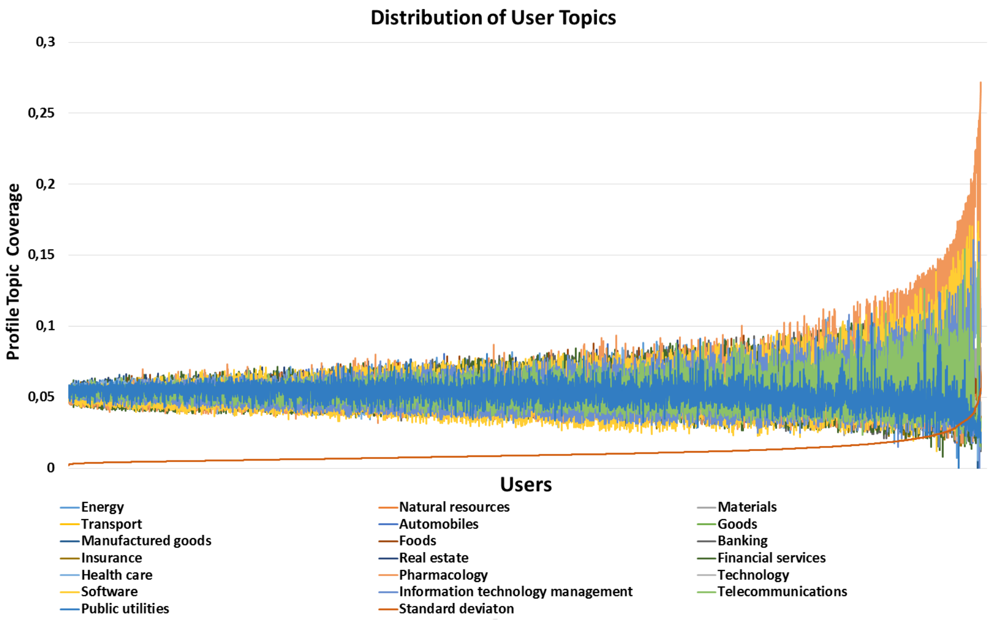


Figure 11: Examples of 3 different users and the corresponding profiles.

Additionally, we analyzed how the topics the users talk about are connected to each other, as shown in Figure 12. The diagram shows the Pearson correlation between the different industries based on the user profiles. We can see some strong correlations within the different sectors. This is expected since the topics of the Industries, such as Hardware and Software, are much related. Interesting to see is that there are also other relations between some of the sectors. For instance, the industries around Healthcare are also connected to the industries around food and chemistry, or the industries around computer hardware show some connections to entertainment. These connections can help finding users which have some domain knowledge in a certain are even when they do not post much about these domains.

Topics Relation to influence

Furthermore, we analyze how this behavior influences the position of users within the social network with respect to their influence (based on followers & lists) and their reputation (based on retweets). For analyzing the top contributors for each sector we choose to select the top 100 users based on the highest percental coverage in that sector within the user profile. For each of those we calculate their average number of followers, friends, statuses and listed counts. The results are shown in Table 7

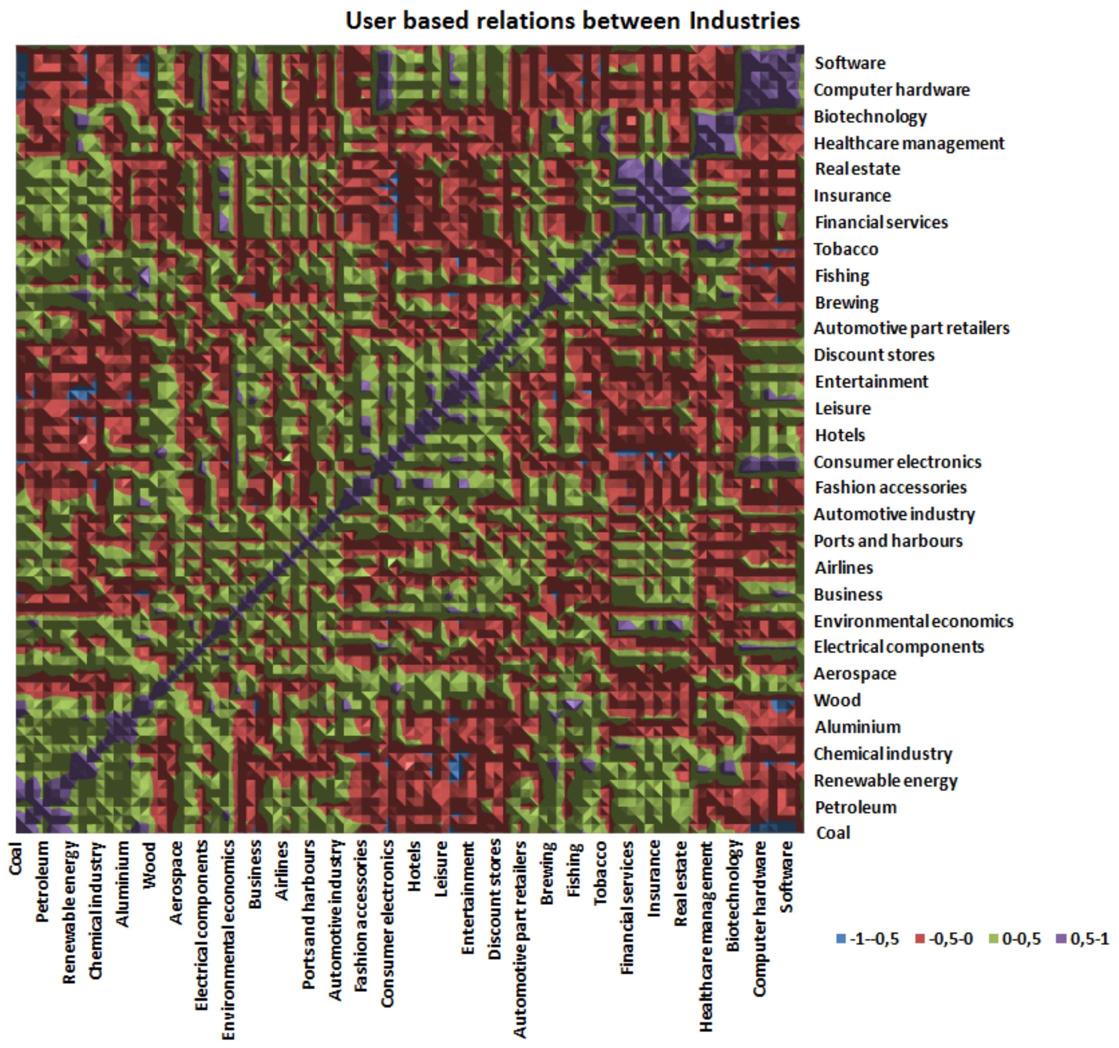


Figure 12: Correlation of different Industries based on User Messages

Experiments for Topic Modelling and Event Detection

In this section we will describe a series of experiments showing how the expert detection methods can improve the effectiveness of event detection described in Section 3.4 or trend modelling in the area of stock market analysis. The main research questions for this experiments are:

- **Q1. Which users talk about which companies?** We can distinguish users based on their calculated interests and posting behavior in certain domains and industry sectors. Based on this, we analyze how effective a selection of certain users for an industrial sector is. And how many of the related messages for this sector we get based on a selection of few expert users.
- **Q2. Are expert users better for event detection and prediction?** Our dataset contains tweets related to stocks, these stocks are related to events. By analyzing the tweets from different user groups we can get different predictions. We analyze the timeframes in which expert users talk about a certain stock and in which time-frame the "normal" users mention this stock.

Table 7: Relation between Influence and Topics

Top 100 Users	Avg Follower	Avg Statuses	Avg Friends	Avg Listed
Energy	2559.94	17243.63	1590.79	65.37
Natural resources	4685.93	17365.61	3040.12	83.96
Materials	5259.67	23385.05	3023.05	66.88
Transport	4583.44	22488.75	1845.28	70.97
Automobiles	2700.87	27310.71	1961.97	28.91
Goods	2800.68	10056.88	1134.21	80.32
Manufactured goods	4300.66	12220.56	2690.9	28.45
Foods	5153.03	24691.78	2047.7	53.06
Banking	4601.98	17592.06	1327.5	125.3
Insurance	4780.14	14988.76	1053.38	136.6
Real estate	2460.12	14024.47	1154.1	51.82
Financial services	2870.38	7825	1126.87	63.59
Health care	3096.98	8971.55	1110.2	110.4
Pharmacology	3173.98	6302.31	597.75	113.59
Technology	4343.17	23063.76	1625.44	129.33
Software	4109.08	19703.43	1393.48	118.25
IT management	3575.85	15880.88	1462.06	124.63
Telecommunications	3894.77	18417.04	1281.52	100.71
Public utilities	1467.21	12218.47	696.52	27.72
Average	3706.2	16513.19	1587.52	83.15

Dataset
For this series of experiments we use same data set as described in Section 3.3.3. Additionally, we generated profiles for a set of 10 different Companies of different sectors. These profiles were generated based on the Wikipedia pages of the corresponding companies and the outlinks of these pages. Two example profiles for the companies “Ford” and “Merck” are shown in Figure 13. We can see a strong focus in the areas of Automobiles and Pharmacology, which are the main areas of interest for these two companies.

For further evaluation we selected 10,000 Users out of the 333,000 Users in our Dataset. These users where randomly selected with the constrain having more than 100 tweets and more than 100 followers, since we want to focus on professional users. For each of the chosen companies we collected the sets of the 100 and 1,000 most similar users based on the Pearson correlation between the generated profiles of users and companies. For our first research question we analyzed the percentages of our user groups which had mentioned one of the analyzed stock names within the monitored period. The results are shown in Table 8. The largest percentages are highlighted. In two of the cases the selected user groups did not match the users talking about the stock so percentage of users within the top users was smaller than the percentage within the set 10k active users. For the other companies the sets of top users contained a higher percentage of users which talked about the company.

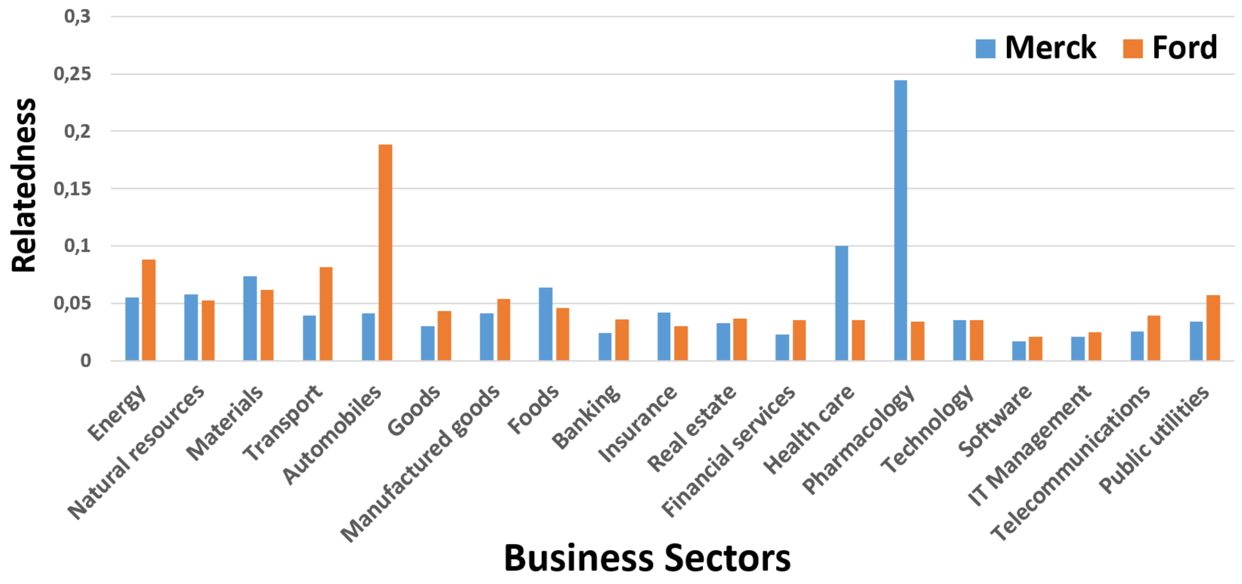


Figure 13: Examples of 2 different companies and the corresponding profiles.

Table 8: Percentage of User Groups Mentioning Stock Names

Company Name	Active Users	Top 1000	Top 100
Apple Inc	63.69	62.1	**81.0**
Amazon	**39.39**	22.6	17.0
Alibaba	22.5	22.6	**35.0**
IBM	14.97	13.8	**19.0**
Yelp	10.07	6.1	**11.0**
Dow Chemical	**4.95**	2.2	0.0
General Electric	9.19	**14.8**	13.0
Ford Motor Company	9.44	6.7	**19.0**
Merck & Co	10.24	33.7	**73**
Pepsico	8.77	2.6	**9.0**
Average	19.32	18.72	**27.7**

Additionally, we collected different events for all companies from Yahoo Finance[3]. We choose events which may have direct influence on the stock market like the announcement of the quarterly reports. Table 9 shows which companies we used and how much content we got for each of the companies within our original dataset.

[3]http://finance.yahoo.com/

Table 9: Statistics about Stock related dataset.

Company Name	Stock - Symbol	tweets	Users
Apple Inc	$AAPL	625,950	110,112
Amazon	$AMZN	152,952	34,618
Alibaba	$BABA	82,733	12,685
IBM	$IBM	27,875	6,018
Yelp	$YELP	10,777	2,379
Dow Chemical	$DOW	4,198	1,194
General Electric	$GE	10,981	2,135
Ford Motor Company	$F	15,308	3,457
Merck & Co	$MRK	12,784	2,810
Pepsico	$PEP	9,321	2,564

Each of the chosen events shows a clear spike regarding the number of tweets containing the stock symbol of the corresponding company. Figure 14 shows one of our example events. We can see that the use of only top 100 expert users also allows to clearly detect the event. This indicates that amount of users we need to monitor can be reduced if necessary. Further analysis is needed to verify how an optimal group of users can be found per stock market sector and how the posting behavior differs.

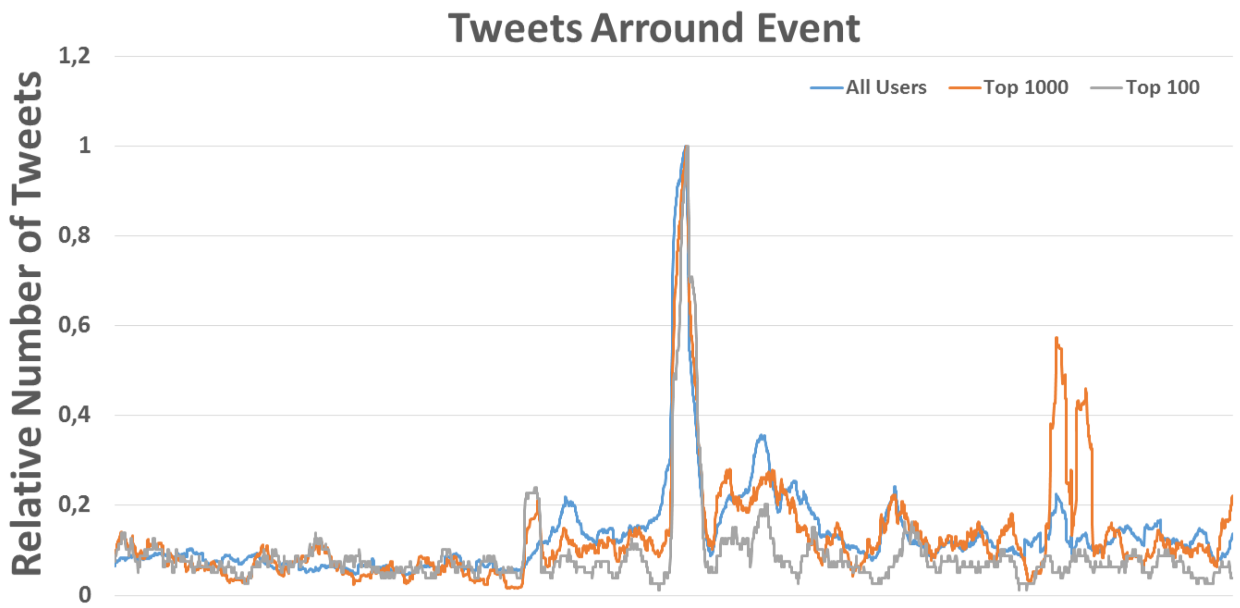


Figure 14: Examples of 2 different companies and the corresponding profiles.

Scalability

In terms of scalability we can assume that the presented methods and algorithms scale very well, since the tweets are processed indecently and later aggregated on a user level, the methods can be parallelized relative easily. The currently largest implementation issue of the method is the large disk space requirement of the annotation tools, which can due to API limitations not be within the HDFS. We are currently investigating different alternatives for the annotations.

Diversity

In terms of diversity the algorithms offers various parameters for increasing the diversity of the chosen users. In general different topics also have different experts based on the presented methods. The overlap between related topics, such as IT management and Software is in the area of 50% while less related areas like Banking and Health Care have no overlap in the top 100 users. Additionally we are investigation how new categories can be found in an automated way, so that a divers set of categories is created, covering all relevant areas with very little overlap.

Discussion & Future Work

The experiments described in this section give an overview about the quality of the user modeling methods and their possible applications. We showed that for the different areas of the stock market experts can be found based on their previous posting behavior. The use of NER methods allow us to interlink different categories and aspects of the stock market based on background information collected from Wikipedia. As a next step we want to generate a larger set of experts for various domains, this would allow us to directly start monitoring a set of users whenever an interesting event occurs. After the final integration of the component into the QualiMaster infrastructure, we also plan to integrate results of this component into the input of other component, such as the adaptive streaming, to see how the results of these components are affected by changing the input based on the detected expert users.

3.4 Event Detection in Social Media

Our system analyzes incoming social media streams in the form of tweets that mention certain stocks or market players and tries to detect stock events using this social media data. This is just one of many possible ways of integrating Event Detection in QualiMaster. It refers to Use Case 7 of D2.1. In this Use Case the system receives social media data from a stream and detects event in the form of bursts or other stream abnormalities. The output consists of the event's time period and additional context information (e.g. keywords). Aside from the use of social media data, stock events can for example also be detected by using news media data or data from public calendars.

With a capacity of more than 500 million tweets per day Twitter is an enormous resource and only a small fraction of the daily tweets deal with stock-related topics. In order to observe the broad perception of certain market players on Twitter, we simply count the times a particular stock has been mentioned in tweets of a time period. In the next step we apply an algorithm of our Event Detection Family to detect bursts in the number of times single market players are mentioned. Bursts can vary strongly in length and strength and different approaches will arrange events differently. In what follows we will take a closer look at the algorithms in our Event Detection Family followed by an evaluation of both the Moving Average Based and the Kleinberg algorithm.

3.4.1 Event Detection Family

Our Event Detection Family comprises two algorithms for Content-based Event Detection for Stock Mentions in Twitter Streams. These algorithms are imbedded in an Apache Storm Topology. In both cases tweets are first preprocessed within a StocksParserBolt. This Bolt receives tweets from an input spout and extracts the stocks that are contained within the content of each Tweet. This information is forwarded to the particular Event Detection Bolt where the event detection takes place. The final output contains information about the extracted events. Each event consists of a time span and a list of keywords associated with the event. The user can decide which stocks and keywords she is looking for. The number of stocks can vary from one to an arbitrary long list of stocks.

The event detection architecture is depicted in 15.

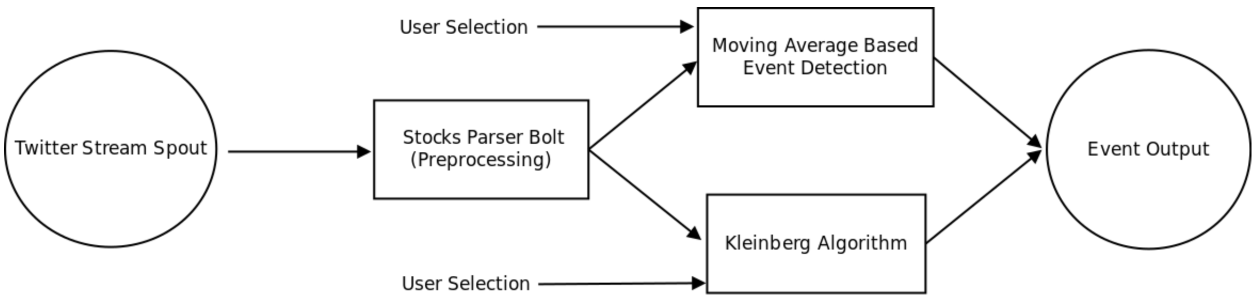


Figure 15: Structure of the event detection topology

3.4.2 Moving Average Based

The Moving Average approach is widely used for analysis of data series. It has been used in the field of social media analysis before [25, 26, 27, 28, 29]. In our system each time a Tweet referencing a stock is detected all relevant tweets of the current time window are regarded and the number of times each stock has been mentioned within this time window are counted. If the value of this stockCount for a particular stock exceeds a certain threshold within the window an event is detected for this stock. The size of the time window is defined by the user using the granularity parameter. The threshold values are based on typical average values for each stock, therefore an event indicates a noticeable deviation from the average number of times a stock is mentioned within a time period of the given size. If several events for a certain stock are detected in a direct sequence, these events are aggregate to a single event of a larger time span. Still, gaps within the sequence will lead to separated events. An advantage of (our implementation of) the Moving Average Algorithm is the fact that there are no static boundaries between the time steps, but instead the boundaries are moving with time. Therefore longer events across boundaries can be detected.

3.4.3 Kleinberg Algorithm

The Kleinberg Algorithm for burst detection has been introduced by Jon Kleinberg in 2003 [30] and has become quite popular [31, 32, 33, 34]. It is based on the assumption that an input stream can be modeled by an infinite-state automaton. Bursts within the input stream appear as state transitions between the states of the automaton. Higher states correspond to higher appearance rates of the item of interest within the input stream. A transition to a higher state demands a certain cost, whereas transitions to lower states are free. In our system the KleinbergBolt collects Tweet data for the period of 10 time steps. For each time step the number of tweets mentioning a particular stock is divided by the total number of incoming tweets mentioning at least one stock. Therefore we can compare the presence of particular stocks in different time steps. If the resulting value is particularly high for one or more time steps compared to the other time steps the Kleinberg algorithm will detect an event. The number of events detected and their particular strength depends on several parameters. The Kleinberg algorithm is less prone to input noise and is able to detect longer bursts as long as they don't cross the boundaries of the considered time period. The Kleinberg algorithm also offers a value for every burst's strength.

3.4.4 Implementation Details

Algorithm 3 describes how the system uses Moving Average Based event detection to detect events within a given set of preprocessed tweets. For the detection several parameters are required: the stock of interest, the size of the time window, the average number of tweets mentioning stock s within a window of size w and a threshold that defines how strong a burst has to be to be considered as an event. The values for s, w, a and t are defined by the user. Of all given tweets only the tweets within the time window containing the stock s are considered. The number of considered tweets is compared to the given average. If this number exceeds the average value by a given threshold an event is detected. In this case the output provides more information about this event.

Algorithm 3: Detecting events with the Moving Average Based Approach

```
Input:
$s$: stock
$T$: preprocessed tweets
$w$: window size
$a$: average for stock s
$t$: threshold
Output: event information
1 begin
2     $W \leftarrow \emptyset$ // Initialization
3     $beginOfWindow \leftarrow determineBeginOfWindow(w)$
4     for $T_i \in T$ do
5         if $date(T_i) > beginOfWindow \ \& \ containsStock(T_i, s)$ then
6             $W.add(T_i)$
7     if $size(W) > a + t$ then
8         return $event(beginOfTimeWindow)$
```

Our system uses the Kleinberg implementation used in [35]. Within the KleinbergBolt we had to adjust the way tweet information is buffered because the Kleinberg algorithm needs the information of several time steps at once to properly detect events within a given time period. We decided to consider 10 time steps at once. The number of time steps may be changed later on for a broader comparison between different time steps or for due to runtime issues.

3.4.5 Evaluation

Dataset

The dataset we used for evaluating the event detection is a subset of the data set we used for evaluating the User and Social Network Analysis as described in section 3.3.3. The focus of the evaluation laid on the market players Apple (AAPL) and Amazon (AMZN) and their presence in tweets in the last quarter of 2014, a timespan including events like the Apple Special Event on 16th of October and the Black Friday on 28th of November as well as more finance-centric events like the Apple Annual Report on 27th of October. Fig. 16 depicts the frequency of tweets mentioning the AAPL stock over time in our test set. We can see that the frequency varies strongly showing bursts on single days. Interestingly these days in most cases do not correlate with major public or finance events of the AAPL stock.

Efficiency

On an average laptop (two 2.53GHz processors, 3.8 GB memory) the processing of 10,000 tweets takes approximately 40 seconds for both the moving average based and the Kleinberg event detection algorithm.

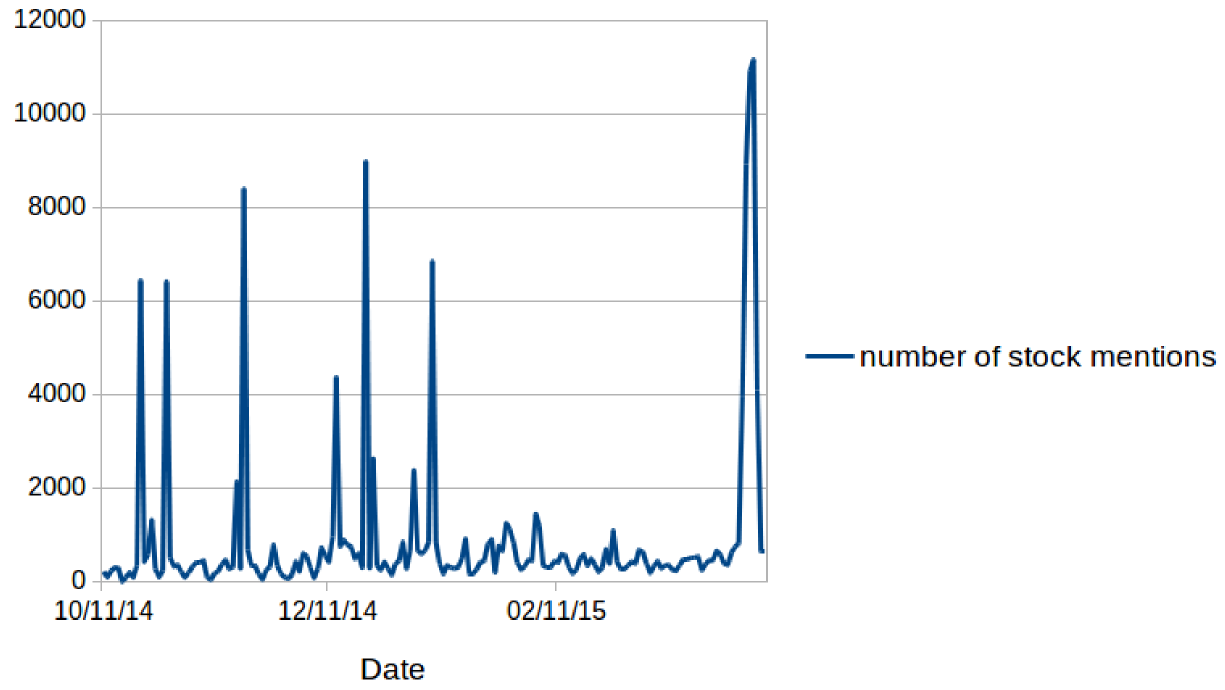


Figure 16: Frequency of tweets mentioning the AAPL stock over time

Quality

For assessing the quality of our approach we compared the events detected by both algorithms to the behavior of the particular stocks within the given time period. We therefore took into account the date and length of each detected event as well as the event's strength for the events detected by the Kleinberg algorithm. The events detected by both algorithms coincide strongly. For the AAPL stock the five strongest events detected by the Kleinberg algorithm were also detected by the Moving Average Based approach while for the AMZN stock 3 of the 4 strongest events matched. Overall the events detected for the AAPL stock were considered to be stronger by the Kleinberg algorithm compared to the events in the AMZN stock. For both stocks the detected events are distributed over the full length of the considered time period with some (but not all) of them matching with time points of conspicuous behavior of the particular stock. The events coincide with the strong frequency bursts in our test dataset which themselves do not coincide with known events like the Apple Special Event, the Black Friday or the Apple Annual Report event though these events seem to have an impact on the stock's development.

Scalability

One of the most important parts of event detection is the aggregation of data and the determination of averages. Although these tasks can be performed in linear runtime, they are typically hard to parallelize.

Diversity

In the case of Event Detection a high diversity correlates with a high number of different sources providing the information we take into account when determining an event. In the special case of Event Detection using Twitter Streams the sources correspond to Twitter users mentioning the particular stocks. In general the number of sources included in the detection of an event provides information about the certainty of the event. A burst induced by only a single user generating lots of tweets within a certain time period should not be considered as an event. Therefore it is important to regard the number of different Twitter users mentioning a stock in their tweets.

17 depicts the number of different Twitter users mentioning the AMZN stock per time step in our test set. The average number of different users mentioning the AMZN stock per day is 21, with some days deviating very strongly from this average. On 24th of December and on 3rd of January more than 1000 different users mentioned the AMZN stock in their tweets. These days correlate with events detected by the Kleinberg algorithm.

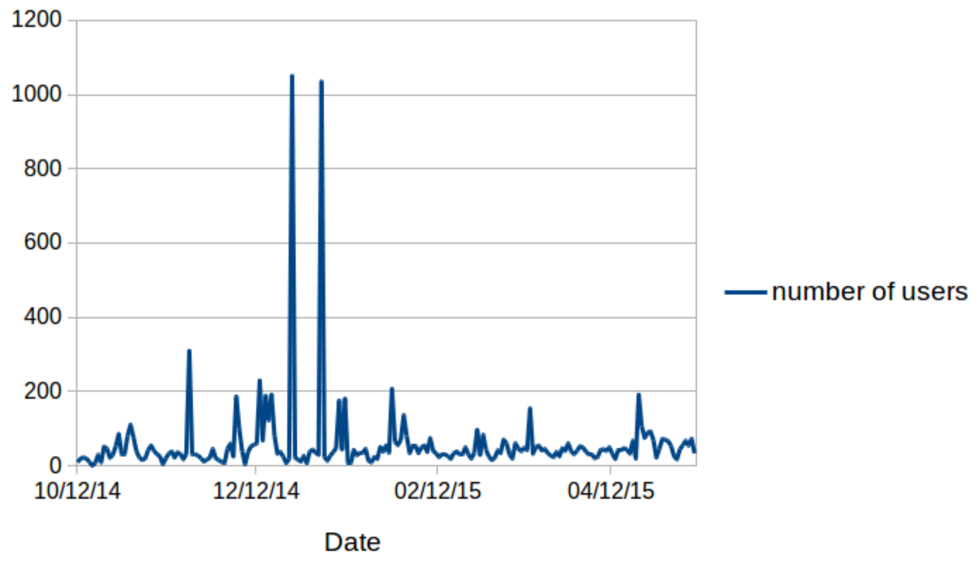


Figure 17: Number of distinct users tweeting about the AMZN stock

3.4.6 Summary

Our approach is able to detect strong increases of stocks mentions in Twitter Streams for arbitrary stocks using two different algorithms, each of them having its own advantages. Both algorithms detect very similar events. For future work we want to include the consideration of keywords to classify different event types.

3.5 Streaming Networks in Social Media

3.5.1 Introduction

Thus in this subsection we describe a method to extract networks of market players based on social data streams. It is of great interest to discover connections and possible dependencies between the market players by the gossiping of the crowd and to compare the result with the opinion of the experts.

In fact, we build on an earlier work [36] about guided pattern mining for extracting large social networks using search engines. Further related research in the area includes (among others) extraction of social connections between fictional characters inferred from dialogs in books [37] or from the narrative of an Ottoman scholar and world traveler [38].

3.5.2 Entity Network Extraction from Social Media

We consider the Social Media to be a stream of data without beginning and end. Given a single unit n in the stream, i.e., a Tweet or a news article, we extract the simultaneously mentioned stock symbols. With this information we are first able to construct a graph, i.e., connect the different market players with each other. The nodes of the graph is the set of all collected symbols. Using the weight per entry w_n which is defined as the reciprocal of the number of stocks n_s in the unit, i.e.,

$$w_n = 1/n_s\,, \tag{3.1}$$

the edges e_i are the accumulated weights per entry over all considered units N

$$e_i = \sum_{n=1}^{N} w_n\,. \tag{3.2}$$

Note that for the visualization purposes we normalize the edges, by the maximal found edge, i.e.,

$$e_i \equiv e_i/\max(e_i) \in (0,1]. \tag{3.3}$$

For a given timespan, to explore the temporal evolution of the interest in particular stocks, we compute the accumulated weight and number count for a sliding window of shorter duration. For example, using a timespan $\Delta T = 90$ days, we can obtain 90 measurements using a sliding window of duration $\Delta t = 1$ day. The average weight in our network was 7.2 with a standard deviation of 1.64.

3.5.3 Entity Network Visualization

The dependencies between the different market players in a static picture are best represented by a graph visualization. In this, the nodes are the sock symbols, and their size is defined by the number count, while the thickness of the edges is given by the accumulated weight. Observing longer timespans with even hundreds of database records yields rather opaque picture, therefore we introduce additional edge weight filter, i.e., we show only edges with weight exceeding certain value. In other words, we filter out unimportant market players.

Due to the nature of the source, for a meaningful visualization of the dynamics of the data, we switch back to the traditional two-dimensional plots of time versus parameter of interest.

An example is given in Figure 18 where a graph is constructed using $AAPL,$FB and $IBM as a seed market players. We can see three big clusters One is containing social networks like Twitter($TWTR) and Facebook($FB) along with some less expectable players like electronic cigarette($ECIG) and imunnotherapie company Advaxis($ADXS). The second cluster includes Microsoft ($MSFT), Cisco($CSCO) and Ali Baba ($BABA). The third cluster includes Netflix($NFLX), IBM($IBM) and Google($GOOGL).

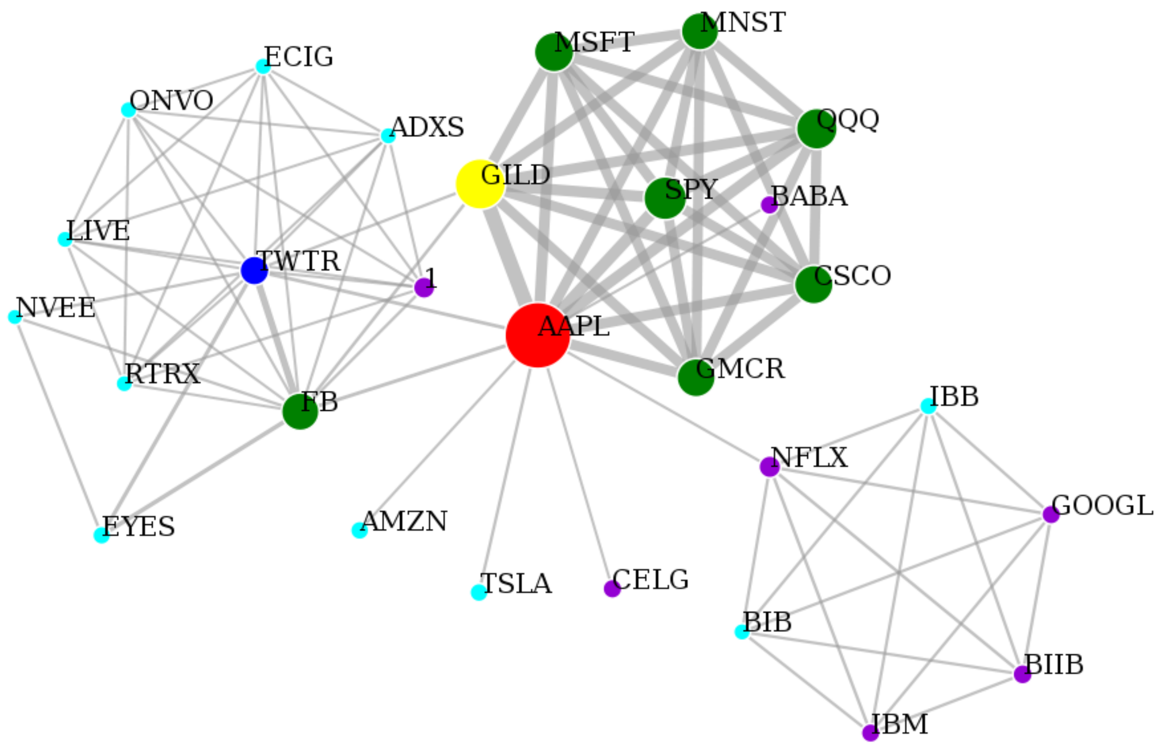


Figure 18: Entity Network visualization for the symbols AAPL, FB and IBM. The timespan equals to three months. The sliding window is of duration one day.

3.5.4 Implementation Details

The Stream Processor is a stand-alone process implemented in Java using the Storm framework. The Storm topology is depicted in Figure 19. It consist of an Input and Output spouts, and Parser and Analyzer bolts. The task of the Parser is to extract stock symbols from the tweets. The Analyzer is used to assign the extracted stocks to the related sectors and industry type, as well as to compute weight per entry. The output of the Stream Processor for each tweet is a DecoratedTweet structure, which is transformed in a database record consisting of the creation time of the tweet, the list of the found stock symbols, the list of the determined sectors, the list of the associated industries and the tweet itself.

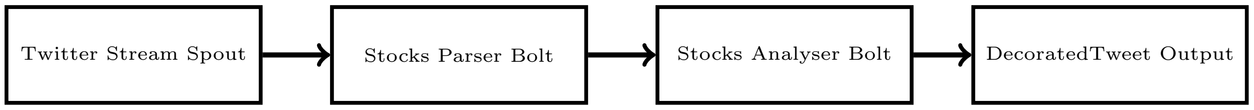


Figure 19: Stream Processor: Storm topology.

To visualize the graph we implemented a software framework depicted in Figure 20. *the Server* is planned in future to receive the data from the HBase storm database. Currently it serves requests coming from the Client, i.e., for a requested time period and filter, the Server constructs the graph representation of the data as found in the database. A pseudo code representation of the algorithm used to construct the graph is given in Algorithm 4.

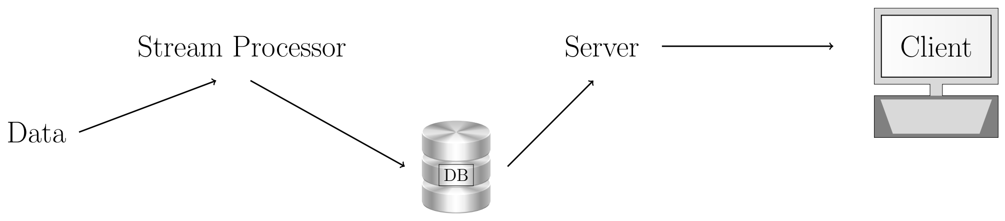


Figure 20: Software units scheme.

In addition upon request the Server is able to compute the temporal evolution of the stocks of interest, in particular, the number count and node weight.

The Client is implemented in HTML, CSS and JavaScript. In particular we use the *dojo* library for the control interface and the *d3* library[4] for the graph visualization and temporal plots. The Client is served by a HTTP server, e.g., Apache and then can be accessed through a Web browser. Given the user defined filter through the control interface, the Client sends request to the Server, which answer in a json formatted blob of data.

[4]http://d3js.org/

Algorithm 4: Graph Construction from a list of Stock co-mentions.

```
Input: $I$: initial list of DB entries
Output: stocks graph $G = (V, E)$
1  begin
2      $E \leftarrow \emptyset$ // Edges
3      $V \leftarrow \emptyset$ // Vertices
4      foreach $I_i \in I$ do
5          $E_i \leftarrow getListOfEdges(I_i)$
6          $V \leftarrow addVertices(I_i)$
7          foreach $e_i \in E_i$ do
8              if $\{e_i\} \in E$ then
9                  $UpdateWeight(E, e_i)$ // update edge weight within the graph
10             else
11                 $E \leftarrow E \cup \{(e_i)\}$
12     return $(V, E)$
```

3.5.5 Evaluation

Dataset

The dataset contains tweets issued within a period of three months, namely from 01.12.2014 to 28.02.2015. Using the public Twitter API we are able to sample 1% of all tweets at most. As we are interested in the financial domain, we do filtered request based on the most interesting 2900 stock symbols. The probability density function for the co-mentions of a stock symbols is shown in Figure 21. We observe that it is most probable to obtain a single stock symbol from a tweet, followed by a group of 4 stocks.

Efficiency

The hot spot in the efficiency of the network extraction is the constructions of the graph from the Server, as we need to compare the new edges build from the nodes of each entry against each other to the already found edges from previous entries. Thus we expect quadratic behavior with the amount of the data that we need to process. To illustrate this, we filter the dataset simultaneously for the three stock symbols AAPL, FB and IBM and measure the response time of the Server for requested number of entries $n_r \in [100, 10000]$ with step $\Delta n_r = 100$. A least square fit to the measured data is plotted in Figure 22. The processing of 10000 tweets takes approximately 40 seconds.

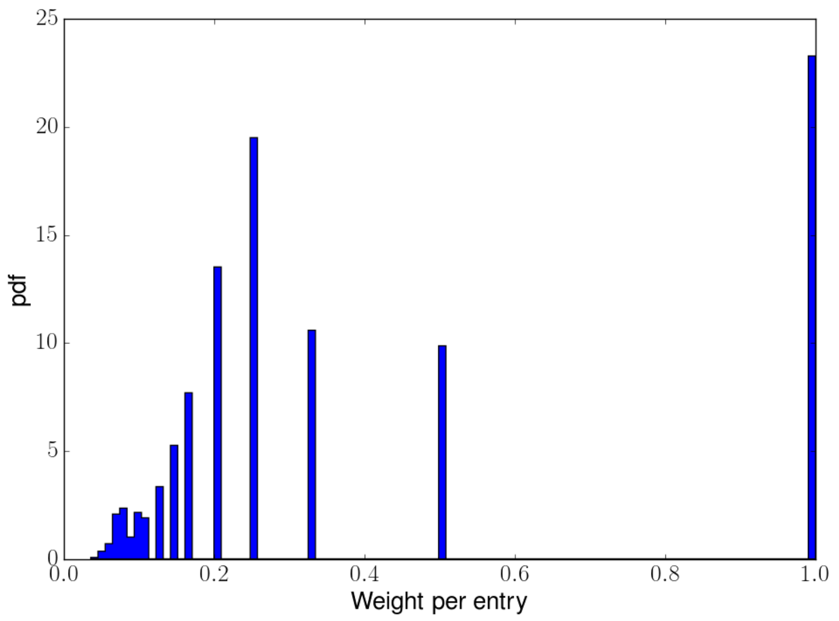


Figure 21: Dataset statistics: pdf of the co-mentions.

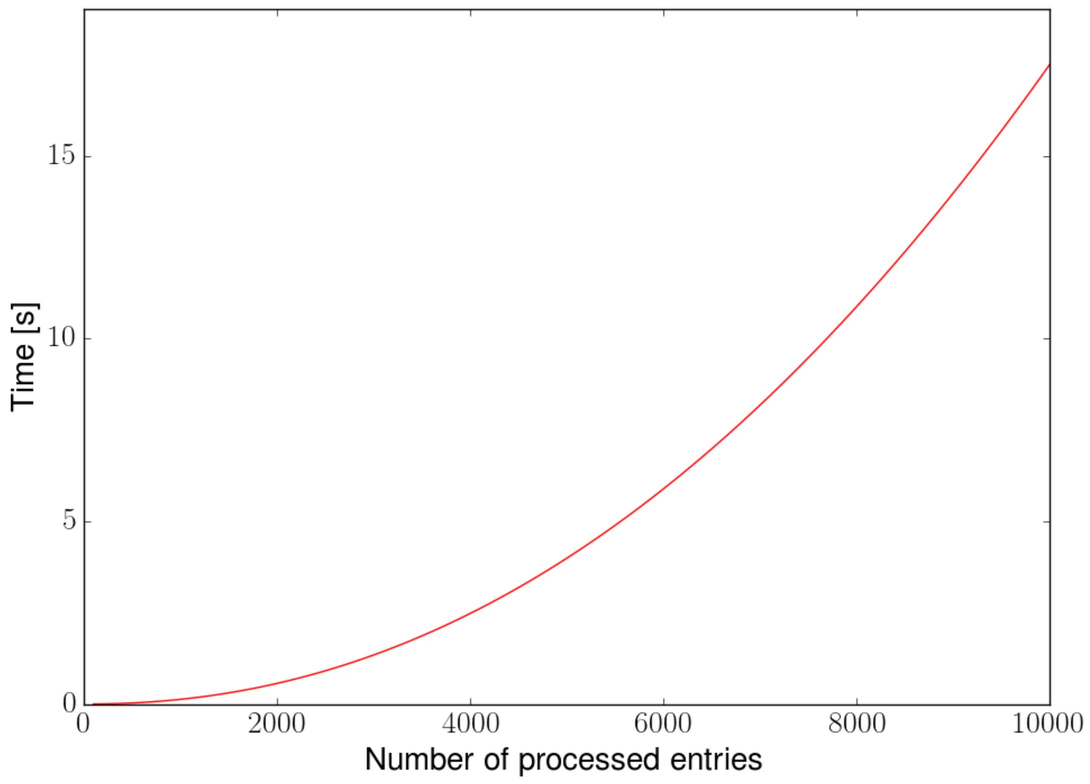


Figure 22: Server throughput.

Quality

To evaluate the accuracy of the extracted networks and the structure of the financial graphs and their properties we intend to carry out a large user study similar to the one done in [36]. In particular, we are going to ask non experts questions targeted towards possible relation between arbitrary market players and requiring answer based on third party source, e.g., a search engine.

Scalability

The Stream Processor is easily scalable through the Storm framework, where the throughput can be increased simply by adding hardware. The challenge in scalability comes however from the server constructing the graph from the data. As long as the graph construction algorithm has quadratic complexity, i.e., $\mathcal{O}(n^2)$, working with millions of database records is prohibitive for real time exploration of the data.

Data Diversity

We use only Twitter as a data source, however the algorithm can be immediately applied to any list of Stocks extracted by the Stream Processor. An interesting candidate in this respect are the StockTweets - the communication platform for the investing community as well as Blogs or News streams.

3.5.6 Summary

The extraction of the financial networks from streaming sources and their dynamics takes central role in the observation of the dependencies between the different market players and is thus inevitable for the risk analysis. There are two main aspects of further development. First, the graph construction algorithm, either through re-implementation or through redesign, should be brought in a state to operate with millions of database records in near real-time. This would allow for longer time spans of observations and better temporal resolution. Second, we can enrich the algorithm with sentiment analysis techniques(e.g. color encoding of sentiment within a cluster), see Section 3.2 and user analysis, see Section 3.3 (only consider edges mentioned by experts), in order to increase the quality of the visualization.

4 Conclusions and Future Work

This deliverable is the second deliverable of WP2. It describes the development state of algorithms and methods in form of pipeline components for scalable and quality-aware real-time data stream processing. Prototypical implementation and first evaluation of the components was carried out within the first half of the second year of the QualiMaster project and reported in this deliverable.

The developed algorithms include processing of financial as well as social web data streams. Computation and appropriate graph query facilities (Section 2.3) and visualization of market player correlations in terms of trading (Section 2.1, 2.2) and mentions in social media (Section 3.5) can provide the financial expert with insights into often uncovered dependencies in the financial market; those algorithms were also identified as extremely useful by the advisory board. Components for sentiment analysis (Section 3.2) help to identify the feelings in the market in different granularity, be it general, player or financial area related. Pre-filtering component (Section 3.1 and user analysis(Section 3.3) lead to focusing on relevant high quality content on the high velocity data streams. Finally, event detection (Section 3.4) as identified by advisory board of the project, is important to keep the expert informed about important events related to selected financial areas, or particular market players. A special focus of all proposed algorithm and their novelty is in the ability to handle large amount of data in form of high velocity streams.

As a proof-of-concept, all of the algorithms have been implemented or adjusted to be executed as Apache Storm (sub) topologies and some of them improved and integrated within WP5 into Priority Pipeline as a first prototype. This includes the Hayashi-Yoshida Correlation Estimator as well as the SVM and SentiWordNet sentiment classifiers.

The next steps in WP2 will be the further development and optimization of the algorithms and other methods presented in D2.1 We will continue conducting extensive experimental evaluations of the quality, performance and scalability of the developed algorithms. Strategies to combine financial and web data streams in a meaningful way in order to support the financial risk analysis application are to be investigated and evaluated. A particular example is to combine network information, or sentiment analysis result. At this point we are currently acquiring and generating test data sets.

References

[1] Takaki Hayashi and Nakahiro Yoshida. On covariance estimation of non-synchronously observed diffusion processes. *Bernoulli*, 11(2):359–379, 04 2005.

[2] Nicolas Champagnat, Madalina Deaconu, Antoine Lejay, Nicolas Navet, and Souhail Boukherouaa. An empirical analysis of heavy-tails behavior of financial data: The case for power laws. July 2013.

[3] Stefan Straetmans and Sajid M Chaudhry. Tail risk and systemic risk of us and eurozone financial institutions in the wake of the global financial crisis. 2013.

[4] Bradley Bensalah and Taqqu Seluk. *Financial risk and heavy tails*. 2003.

[5] Shiraj Khan, Sharba Bandyopadhyay, Auroop R. Ganguly, Sunil Saigal, David J. Erickson, Vladimir Protopopescu, and George Ostrouchov. Relative performance of mutual information estimation methods for quantifying the dependence among short and noisy data. *Phys. Rev. E*, 76:026209, Aug 2007.

[6] Taiji Suzuki, Masashi Sugiyama, Jun Sese, and Takafumi Kanamori. Approximating mutual information by maximum likelihood density ratio estimation. In *FSDM, held at ECML-PKDD*, pages 5–20, 2008.

[7] Thomas Schreiber. Measuring information transfer. *Phys. Rev. Lett.*, 85:461–464, Jul 2000.

[8] Lee J, Nemati S, Silva I, Edwards BA, Butler JP, and Malhotra A. Transfer entropy estimation and directional coupling change detection in biomedical time series. *Biomedical engineering online*, 11:1–17, 2012.

[9] Yunyue Zhu and Dennis Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 358–369. VLDB Endowment, 2002.

[10] European Central Bank. Recent advances in modeling systemic risk using network analysis. 2010.

[11] Monica Billio, Andrew Lo, and Loriana Pelizzon. Econometric measures of connectedness and systemic risk in the finance and insurance sectors. *Journal of Financial Economics*, 104:535–559, 2012.

[12] Delphine Lautier and Franck Raynaud. Systemic risk and complex systems: A graph-theory analysis. In Frdric Abergel, Bikas K. Chakrabarti, Anirban Chakraborti, and Asim Ghosh, editors, *Econophysics of Systemic Risk and Network Dynamics*, New Economic Windows, pages 19–37. Springer Milan, 2013.

[13] Tom C. W Lin. Too big to fail, too blind to see. 2010.

[14] Odysseas Papapetrou, Minos N. Garofalakis, and Antonios Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *PVLDB*, 5(10):992–1003, 2012.

[15] Ravi Kant Jain. Putting volatility to work. *Active Trader Magazine*, 2001.

[16] Dmitry Davidov, Oren Tsur, and Ari Rappoport. Enhanced sentiment learning using twitter hashtags and smileys. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 241–249, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[17] Alexandra Olteanu, Sarah Vieweg, and Carlos Castillo. What to expect when the unexpected happens: Social media communications across crises. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work &#38; Social Computing*, CSCW '15, pages 994–1009, New York, NY, USA, 2015. ACM.

[18] Xiaolong Wang, Furu Wei, Xiaohua Liu, Ming Zhou, and Ming Zhang. Topic sentiment analysis in twitter: A graph-based hashtag sentiment classification approach. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 1031–1040, New York, NY, USA, 2011. ACM.

[19] AlexHai Wang. Machine learning for the detection of spam in twitter networks. In MohammadS. Obaidat, GeorgeA. Tsihrintzis, and Joaquim Filipe, editors, *e-Business and Telecommunications*, volume 222 of *Communications in Computer and Information Science*, pages 319–333. Springer Berlin Heidelberg, 2012.

[20] Surendra Sedhai and Aixin Sun. Hspam14: A collection of 14 million tweets for hashtag-oriented spam research. In *The 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '15*. 2015.

[21] Kilem Gwet. *Handbook of Inter-Rater Reliability*. Advanced Analytics, LLC, second edition, 2010.

[22] David Milne and Ian H Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 509–518. ACM, 2008.

[23] David Milne and Ian H Witten. An open-source toolkit for mining wikipedia. *Artificial Intelligence*, 2012.

[24] Rudi L Cilibrasi and Paul Vitanyi. The google similarity distance. *Knowledge and Data Engineering, IEEE Transactions on*, 19(3):370–383, 2007.

[25] Junghoon Chae, Dennis Thom, Harald Bosch, Yun Jang, Ross Maciejewski, David S Ebert, and Thomas Ertl. Spatiotemporal social media analytics for abnormal event detection and examination using seasonal-trend decomposition. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 143–152. IEEE, 2012.

[26] Daniel Gayo-Avello. Don't turn social media into another'literary digest'poll. *Communications of the ACM*, 54(10):121–128, 2011.

[27] Nicholas A Diakopoulos and David A Shamma. Characterizing debate performance via aggregated twitter sentiment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1195–1198. ACM, 2010.

[28] Vasileios Lampos and Nello Cristianini. Tracking the flu pandemic by monitoring the social web. In *Cognitive Information Processing (CIP), 2010 2nd International Workshop on*, pages 411–416. IEEE, 2010.

[29] H Russell Fogler and Fred Nutt. A note on social responsibility and stock valuation. *Academy of Management Journal*, 18(1):155–160, 1975.

[30] Jon Kleinberg. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery*, 7(4):373–397, 2003.

[31] Ravi Kumar, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. On the bursty evolution of blogspace. *World Wide Web*, 8(2):159–178, 2005.

[32] Dan He and D Stott Parker. Topic dynamics: an alternative model of bursts in streams of topics. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 443–452. ACM, 2010.

[33] Qiming Diao, Jing Jiang, Feida Zhu, and Ee-Peng Lim. Finding bursty topics from microblogs. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 536–544. Association for Computational Linguistics, 2012.

[34] Chaomei Chen, Fidelia Ibekwe-SanJuan, and Jianhua Hou. The structure and dynamics of cocitation clusters: A multiple-perspective cocitation analysis. *Journal of the American Society for Information Science and Technology*, 61(7):1386–1409, 2010.

[35] Diana Maynard, Gerhard Gossen, Adam Funk, and Marco Fisichella. Should i care about your opinion? detection of opinion interestingness and dynamics in social media. *Future Internet*, 6(3):457–481, 2014.

[36] S. Siersdorfer, H. Ackermann, P. Kemkes, and S. Zerr. Who with whom and how? guided pattern mining for extracting large social networks using search engines. In *CIKM*, 2015.

[37] David K. Elson, Nicholas Dames, and Kathleen R. McKeown. Extracting social networks from literary fiction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 138–147. Association for Computational Linguistics, 2010.

[38] Ceyhun Karbeyaz, Ethem. Can, Fazli Can, and Mehmet Kalpakli. A content-based social network study of evliy elebis seyahatnme-bitlis section. In *Computer and Information Sciences II*. Springer, 2012.