University of Zurich
Department of Informatics

Dynamic and Distributed
Information Systems

Evaluation of Quality Assurance Mechanisms in Paid Crowdsourcing

Bachelor Thesis February 3, 2013

Patrick Winzenried
of Belp, BE

Student-ID: 09-706-961
patrick.winzenried@uzh.ch

Advisor: **Patrick Minder**
Prof. Abraham Bernstein, PhD
Department of Informatics
University of Zurich
http://www.ifi.uzh.ch/ddis

Acknowledgements

It would probably not have been possible to complete this work without the support of following people I would like to acknowledge.

My greatest thanks go to my supervisor Patrick Minder for his continuous and outstanding support. He provided invaluable input and showed me how to address issues and challenges. Also the sheer amount of time he took for me cannot be taken for granted.

I also would like to thank Prof. Bernstein for the introduction of *Crowdsourcing* in his lecture *Distributed Systems*. This was a crucial factor, since I did not only do my bachelor thesis in this field but also my in-depth study during the bachelor degree. This study gave me the chance to get a feeling about the possibilities of crowdsourcing and social operating systems.

In addition to this, I would like to thank my girlfriend for her leniency and her understanding.

Abstract

Computer systems still struggle with tasks such as categorizing photos, translating phrases or verifying collected data. Crowdsourcing platforms like Amazon's Mechanical Turk are the ideal place to solve such challenging assignments by using the collective power of human beings, which are willing to work for a small amount of money. However, not every individual provides perfect answers – some of them try to maximize their income by cheating, while others may misunderstood the task or do not have sufficient skills to solve it properly. In order to hold a certain level of quality, it is necessary to apply some sort of quality assurance mechanisms. This thesis will compare two such mechanisms by applying them on three different use cases and finally presents results about the performance in terms of quality, time and costs.

Zusammenfassung

Computersysteme sind gegenwärtig nicht in der Lage kognitive Herausforderungen wie das Kategorisieren von Fotos, das Übersetzen von Texten oder das Verifizieren von Daten zufriedendstellend zu meistern. Crowdsourcing Plattformen wie Amazon's Mechanical Turk scheinen daher die ideale Lösung zu sein. Dabei wird die kollektive Leistung derjenigen Personen benutzt, die bereit sind, die Aufträge für einen kleinen Geldbetrag zu lösen. Jedoch wird nicht jede Aufgabe perfekt gelöst. Einige Teilnehmer versuchen zu betrügen, andere haben die Aufgabenstellung missverstanden oder es fehlt ihnen das nötige Wissen. Für solche Systeme ist ein gewisses Qualitätsniveau von essentieller Bedeutung, weshalb man Qualitätskontrollen implementieren muss. Diese Arbeit vergleicht zwei solche Kontrollmechanismen unter Anwendung auf drei verschiedene Fallbeispiele und präsentiert schlussendlich die Resultate der Performanz in Bezug auf Qualität, Zeit und Kosten.

Table of Contents

1

Introduction

With the beginning of the computer era more than 60 years ago [Rojas, 1983], almost every area of our life is in one or another way influenced by information technology. It has changed the way of how we communicate, how we search and retrieve information, how we work and how we interact socially.

Business and organizations have also faced dramatic changes. Databases store information about customers and critical data. Mainframes provide almost infinite computing time and especially electronic data processing enabled businesses to focus more on their core products and to operate more efficient.

The introduction of networks, distributed systems and the Internet followed, what makes it easier than ever before to exchange data and information. Companies can now operate globally and individuals are no longer bound to a place or a desk. Servers provide services 24/7, so customers can check their balance online, order products or book their next vacation.

No matter whether it is about creating a document, a piece of software, calculating complex computations or hosting a web service, as long as the central processing unit can follow clear instructions, a computer is the ideal machine for creating, processing, storing and reproducing data. However, there are still areas where a computer does not provide outstanding performance and quality.

When it comes to tasks, where data is ambiguous or the semantics are not clearly determined it gets difficult for a machine to understand the context and to provide satisfying solutions. For example, it is almost impossible for a computer or software respectively to identify all objects within a photograph. Even when human beings would enter via an interface what they see on the picture, there is a high chance that the computer would collect multiple meanings for the same object, which is still very hard for a machine to understand. As another example, there are also limitations when a computer should translate a text into another language. A machine is incredible fast when the task is a classic look-up of a translation of a given word, but when the machine should translate an entire phrase, it gets difficult since it needs to understand the context in order to pick the right word and it should also be aware of the syntax in the target language.

In such cases, human beings still provide better results. They can use computer-supported translation and other tools to improve their working speed, but in the end they are still very slow. There is an approach to this which use the advantages of both worlds: distributed human beings executing over the Internet very small tasks, for example, translating a phrase and software take care of the management in order to make sure that the output is aggregated and sorted. This kind of collaboration is called *Crowdsourcing*.

1.1 Research Question and Hypotheses

However, quality management is a challenging part - crowdsourcing means that one has no truly knowledge about the experience and the skills of a specific crowd worker.
One find itself in a situation where it is uncertain whether a worker provides reliable and truthful answers. Thus, it is difficult to predict the quality of the output, although, keeping a certain level of quality is critical for most tasks. This motivated several researchers to develop quality assurance mechanisms in order to achieve a certain level of quality. For example by using an output agreement or a voting contest. Consequently, this leads to the following research question:

Does one of these quality assurance mechanisms provide better performance than others no matter what the underlying task is about?

Derived from this research question, the author formulated following hypotheses:

1. The *Six Sigma* filtering, outperforms the *Gold Standard* method in terms of quality, time and costs.
2. The *Six Sigma* and *Gold Standard* filtering mechanisms always provide better quality compared against a base line of unfiltered, raw data.
3. The underlying task does not influence the performance of a quality assurance mechanism in terms of quality.

1.2 Contributions

This thesis analyzes collected data sets of three conducted experiments in order to formulate statements about the performance of two quality assurance mechanisms in the context of human computation. These two mechanisms – one filters output based on an average working time method and the other uses a control question technique for output filtering – are combined with a set of aggregation methods and are compared against a bas eline of raw data sets.
The statistical evaluated performance – measured in terms of quality, time and costs – shall help to

understand the strengths of these quality assurance mechanisms and could be a practical guideline for choosing the right approach depending on a requester's priorities.

1.3 Outline

The remainder of this paper is structured as follows. Section 2 is dedicated to related work. In section 3, the quality assurance mechanisms are introduced, whereas the focus lies on the filtering and aggregation mechanisms. The experimental setup and the use cases are explained in section 4. The results of the evaluation are presented in section 5. In section 6 and 7, the thesis is concluded by the discussion about the results, the limitations and the future work.

2

Related work

As soon as one would enter the internet, there is a reasonable chance that one encounters collective intelligence. This may be in the form of a knowledge collection such as Wikipedia as a famous example, where users create knowledge articles in collaboration, or Youtube, where users upload their own content. Even if one is using a search engine like Google, he is benefiting from collective intelligence since Google relies on the recommendations users gave when they link to another information source.

There are already hundreds of examples of collective intelligence. The authors of the paper *The Collective Intelligence Genome* [Malone et al., 2010] collected for their work almost 250 collective intelligence systems. Some of them try to solve scientific problems such as finding *well-folded proteins* (http://fold.it), others encourage users to provide innovative solutions for problems in a broad range of fields such as economics, engineering or physics (http://www.innocentive.com). There are even some examples where the use of collective intelligence is part of the business model. For instance, Threadless (http//:www.threadless.com) where users can upload their t-shirt designs or vote for the best design on a weekly basis. Threadless then produces and sells the shirts and the winning designer gets a financial award.

Beside of these systems, where the tasks are tightly coupled to a specific area, there are also marketplaces such as Amazon's Mechanical Turk (http://www.mturk.com) where a potential requester can access the workforce of thousands of human workers and let them solve tasks such as translations or image labeling for a fraction of the cost of a real employee.

Despite these well-functioning examples, the research of collective intelligence is still a quite new and unexplored scientific field. There is still a lack of knowledge about how to build and use such systems in a well-defined way and most of the available systems fail or are the result of a trial-and-error approach [Bernstein et al., 2012]. In addition to this, there are various definitions and multiple terms such as crowdsourcing, collective intelligence, human computation, crowd wisdom or wikinomics [Doan et al., 2011], where some research articles use them as synonyms [Doan et al., 2011] while others formulate characteristics to distinguish them [Malone et al., 2010].

This chapter will provide an overview of the current state of research in the fields of crowdsourcing, collective intelligence and human computation with a special attention to the definition

of *social operating systems* and ongoing research about quality assurance mechanisms in the context of collective intelligence systems.

2.1 Crowdsourcing, Collective Intelligence and Human Computation

To get an understanding of the terms crowdsourcing, collective intelligence and social computing, a definition for each of them must be provided. How they are linked and why they should not be used as synonyms will be explored in a next step.

Crowdsourcing was first used by the author Jeff Howe in a Wired magazine article [Howe, 2006b]. The link to the term *outsourcing* is quite obvious when we read his definition:

> *"Simply defined, crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call. "* [Howe, 2006a]

This definition leads to the assumption that primarily companies are interested in crowdsourcing. This is understandable as crowdsourcing is an ideal opportunity to include people from all over the world with their experience and knowledge who, otherwise, may not have been possible to do due to geographical reasons – and all that at low costs. Brabham emphasizes this statement in his paper *Crowdsourcing as a Model for Problem Solving* [Brabham, 2008] in which he analyzes several business models. Of course, there may not always be a profitable idea behind every crowdsourcing application – research teams or private persons could also use the power of these large worker pools for their purposes.
Whether Wikipedia, for example, is considered as a crowdsourcing system depends on the literature. Some researchers entitle it as an example [Doan et al., 2011], while others claim it should not be considered as a crowdsourcing system [Quinn and Bederson, 2011].

In a subsequent step, *social computing* should now be defined. According to Quinn and Bederson, blogs, wikis and online communities could be grouped under the term *social computing* [Quinn and Bederson, 2011]. One of the main differences between human computation and social computing is the fact that social computing is not primarily the idea of performing a specific problem oriented task – it is rather social interaction that gets supported by modern technology. Of course, the output of such a social computing platform can also contain the knowledge of their participants – the best example would be a wiki – therefore social computing systems fit into the definition of collective intelligence.

As a final step the definition of *Human Computation* follows:
The modern usage of *Human Computation* was introduced in 2005 by Luis von Ahn's dissertation with the title "Human Computation" which he also published as a book [Law and von Ahn, 2011]. He explained various techniques and ideas with examples like ReCAPTCHA or a game called

ESPGame to show how it is possible to motivate users to perform a simple task. The output of these tasks helps digitalizing books or labeling images. Von Ahn also provided a definition of human computation:

> *"...a paradigm for utilizing human processing power to solve problems that computers cannot yet solve."* [Law and von Ahn, 2011]

Regardless of whether we want to define boundaries for the terms of crowdsourcing, collective intelligence and human computation or that we use them as synonyms, there are always human beings involved who perform a task or generate output that a computer cannot yet perform as von Ahn pointed out above.

However, it is advisable to understand what the differences between crowdsourcing, collective intelligence and social computing are and how to classify them. The authors Quinn and Bederson even consider data mining, which has a peripheral position in the context of collective intelligence. According to them, there are overlaps between these terms, where crowdsourcing and social computing are part of the collective intelligence definition as already set out in the previous definitions. However, human computation must not fit entirely into this scope. Human computation can also occur when a worker solves an assignment isolated and the output is not aggregated in any form with other results. Consequently, there is no collective intelligence [Quinn and Bederson, 2011]. The venn diagram in figure 2.1 illustrates how the different kinds of systems are linked to one another.

Motivation

Because of the increasing participation rates of several such human-based electronic services, for example, the growing worker pool on Amazon's platform Mechanical Turk, it is important to understand by what kind of motivation these human workers are driven. In the case of Mechanical Turk, the motivation is monetarist-oriented. Workers are willing to invest their time and solve the published tasks for a financial reward. Requesters pay for their HITs (human intelligence tasks) on average something between $0.01 and $1, which leads to an hourly average wage of $1.66 as calculated by Paolacci et al. [Paolacci et al., 2010]. As soon as the crowdsourcing system is part of a greater business model, the participants often get paid in money for their effort. In some cases, only the winner gets a financial reward (e.g. Threadless.com, InnoCentive.com).

[Malone et al., 2010] analyzed several other collective intelligence systems where the participants were driven by other kinds of motivation:

Money gene whenever the interest of the participants is not particularly coupled to the output of the company, a financial reward is the easiest way of motivation.

Love gene People participate in a system even when there is no promise of a financial gain. The authors mention several variants of the Love gene, such as —*enjoyment* when playing an online game, *socializing with others* or *contributing to a cause larger than themselves*.

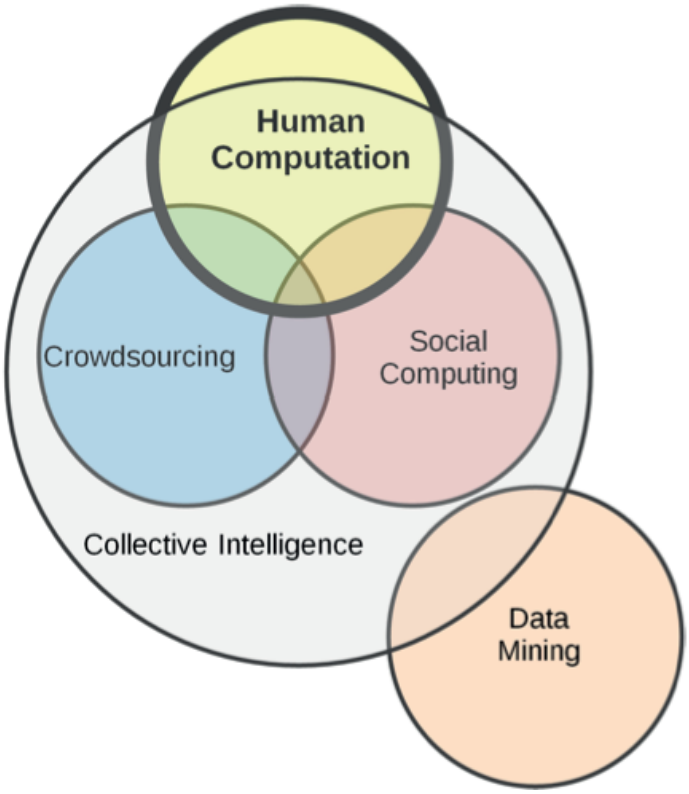


Figure 2.1: Venn diagram illustration of human-based electronic systems [Quinn and Bederson, 2011]

.

Glory gene as a third important motivator they found the Glory gene. To get recognized by others for the own work is a powerful motivation – often seen by programmers who contribute a piece of software for free or provide new features and improvements for a system, for example, the Linux operating system.

In conclusion, it can be said that there is a clear demarcation between intrinsic and extrinsic motivational behavior – highly dependable on the task itself.

2.2 A Social Operating System

The merging of common computer systems with the incredible cognitive power of human beings causes new challenges and requires new approaches in terms of design patterns and allocation of resources. It is necessary that the understanding of a human-based system goes towards a new kind of operating system which should be called a *social operating system* [Bernstein et al., 2012]. A social operating system does not only manage CPU time, memory access and disk space but has also to care about human resources. Not only in terms of time and costs but also in terms of

cognitive advantages of each worker. The social operating system should be able to match a task to the best available human workers in order to guarantee a certain level of quality output.

Bernstein et al. [Bernstein et al., 2012] see for demonstrable reasons certain disadvantages in conventional programming languages and propose therefore the development of a new programming language that would meet those new requirements. Depending on the problem description, the system shall be capable of executing the best approach and has to control, at any given time, constraints like time, costs and quality by setting incentives. In some cases it may have to assign micro-tasks to workers along a large workflow, in other cases it only defines a goal and lets the workers choose how they want to manage the situation [Bernstein et al., 2012].

Currently, one finds on popular crowdsourcing marketplaces only relatively simple tasks such as image labeling, tagging and item categorization [Minder and Bernstein, 2011]. These tasks do not use the entire capacity of the collective intelligence, but only advantage of massive parallel human computation – probably because we are still far away from a social operating system. It is therefore very important that research such as *CrowdLang* [Minder and Bernstein, 2011] is promoted. Minder and Bernstein present a concept of a programming language and a framework which uses existing patterns and recombine them in order to program *human computers* [Minder and Bernstein, 2011]. This means that a problem should be divided into smaller sub-problems and then arranged along a solution path by using different kinds of operators. Such operators could be, for example, a *Divide-and-Conquer* and *Aggregate* method. The problem is split into several sub-problems by the *Divide-and-Conquer* operator and the results are then merged by an *aggregate* operator into a global solution for the actual problem [Minder and Bernstein, 2011]. Minder and Bernstein formulated, in addition to these operators, several others in order to manage a wide variety of sophisticated problems – there are operators for iterations or contests so that the output of a task could be refined in a subsequent step by another worker or it could be voted for the best solution among a set of given task outputs.

2.3 Quality Assurance Mechanisms

With the right incentives, it is possible to convince enough human workers to participate in a problem solving task, but – especially with the prospect of financial gain – there are not only honest workers. There will always be individuals who try to play out the system in order to increase their income. Some of them may just spam by providing random input, others may seek the path of least resistance and provide the least effort to get the assignment done and there are even workers who could put too much effort into a task, which then could have bad impacts on constraints such as time and costs. However, that doesn't necessarily mean that workers have to be experts or trained in order to get the best possible results or to sensitize them, but it is essential that some quality assurance mechanisms are implemented so that a constant level of quality can be assured at all time.

Since tasks can be solved thanks to crowdsourcing in massive parallelization, the most used quality control methods are probably *output agreement* and *filtering*. Based on the huge amount of received answers, one must just select the best output by majority voting or filtering out bad answers by applying a control method such as statistical filtering – Minder and Bernstein applied among other control mechanisms a method called *Six Sigma Pruning* [Minder and Bernstein, 2012]. They monitored the working time for each output and calculated the average working time ω as well as an interval of $\omega \pm 3\sigma$. They were then able to filter out *lazy turkers* (workers who provide bad answers by cheating) and *eager beavers* (workers who put too much effort into the task) which leads to a lower variability of answers [Minder and Bernstein, 2012; Bernstein et al., 2010]. Another filtering method would be *gold questions*: a task contains one or more control questions known as *gold questions* and if a worker provides an acceptable answer for such a gold question (the true answer for this question is known), he probably will also provide acceptable answers for the rest of the task [Sun and Dance, 2012].
The assurance of quality by either statistical methods or decision functions are addressed by several others crowdsourcing researchers as well:
Kern et al. provide in their paper *Statistical Quality Control for Human-Based Electronic Services* a quality control approach based on *Acceptance Sampling*. Based on the quality of a sample, it will be decided whether an entire set of answers gets accepted – this reduces the quality control effort since only sub sets of answers are validated rather than every single output [Kern et al., 2010b]. In order to guarantee a constant level of quality, the acceptance sampling plan starts with a 100 percent full inspection and then switches to sample inspections if the inspection found less defective output than a given threshold. If the quality control detects in a sample a defective unit, it gets back to 100 percent full inspection [Kern et al., 2010b]. This model can also be applied at worker level: a worker who stays continuously in full inspection generates high quality control costs and should be removed from the worker pool for this specific task.
Kern et al. also developed, in another paper, *Quality Assurance for Human-Based Electronic Services: A Decision Matrix for Choosing the Right Approach*, a decision matrix. This matrix helps to choose the right approach depending of the required level of quality and whether the task is deterministic or not. They recommend, for example, the use of a *Majority Vote* when the execution and validation effort are similar and the required level of quality is either high or low and a *Majority vote with improving review* approach for a non-deterministic task where the level of quality must be high [Kern et al., 2010a].
Malone et al. formulated not only money, love and glory genes for motivation but also several other genes for creating tasks and deciding how to handle the output [Malone et al., 2010]. These genes and a short description for each of them are listed in the following overview:

Create

Collection gene task is split into sub-tasks and outputs get collected (e.g. all articles at Wikipedia)

Contest gene useful when only a few or one output of high quality is needed

Collaboration gene useful when task cannot be divided into sub-tasks

Decide

Group decision gene everyone in the group has the same set of decision rules and must apply them to a task or an output

Voting gene useful to get rid of bad output – good output will be voted up

Averaging gene useful when the task is about estimating a number

Consensus gene applicable for small groups – must find a consensus

Prediction Market gene use the *wisdom of the crowd* – useful when estimating a number

Individual Decisions gene individuals can make their own decision

Market gene useful when motivation is driven by money

Social Network gene exchange of knowledge and opinions in order to make own decisions

Sun and Dance [2012] set up an experiment with three more sophisticated methods – using tournament and elimination principles in order to show how a task can be accomplished when Majority Voting fails, which also leads to more economical results.
The *Tournament Selection* draws n items from a pool of size P and lets a person vote for the best answer – this item is then moved into a pool of *next generation* answers. This is repeated until one of the items occurs in the new pool for a certain percentage – this majority item is selected as best item [Sun and Dance, 2012]. As a second method, they test the *Elimination Selection*, where n items are selected from a pool P and a person votes for the best answer. All other items get a *loss* recorded. If an item loses T times, it gets removed from the current pool. This is repeated until the pool contains a specific amount of answers. All remaining items are considered as best answers [Sun and Dance, 2012].
Both of these two methods were compared against the output of *Condorcet Voting* as a base line and the researchers could demonstrate in a Chinese-Idiom experiment that these two methods outperformed the simple voting approach in every setting.
Ipeirotis et al. [2010] also presented a more sophisticated procedure to estimate worker quality by separating error from bias. For illustration, consider a task where a worker has to check whether a specific website has adult content. However, the worker declares every single website as an adult site. The requester then evaluates the output and sees that this worker has an error rate of 50% (in fact every second website contained adult material) – but this is not an accurate conclusion. The worker is just a spammer and always gives the same answer - so he should be removed from further tasks rather than giving him a 50% accuracy [Ipeirotis et al., 2010].

As a final statement in this section, it is important to mention that most of the discussed quality assurance mechanisms are applied ex post – a quality control method determines the quality of the output and has then to decide whether an answer should be accepted. Ipeirotis and Horton [2011] however draw attention to the need for standardization as we know it, for example, from

the production of physical goods. They emphasize the importance of standardized work types in order to generate better requests ex ante on platforms such as Mechanical Turk [Ipeirotis and Horton, 2011]. This would lead to a situation where simple tasks could be traded as in a stock market - resulting in lower costs and a more efficient market [Ipeirotis and Horton, 2011].

3

Quality Assurance Mechanisms

This chapter focuses on the description of the methods that were used in this work. The application of these methods as part of the use cases will be discussed in chapter 4.

The implementation of *Six Sigma* filtering method as well as the method of *Gold Standard* filtering – both combined with mathematical decision methods such as *average*, *median*, *mode* or *output agreement approach* and *voting* in specific settings – shall answer whether the *Six Sigma* method outperforms the *Gold Standard* filtering approach in terms of quality, time and costs regardless of the underlying task. For a subsequent evaluation of both methods, the plain data sets in combination with the decision methods will be used as a base line.

3.1 Filtering Methods

A filtering method can be compared with a predefined mesh size of a net. Only items which are smaller than the mesh size can pass the net, while all other items get sorted out as they do not match the predefined criteria. A filtering method in the context of quality assurance checks for every item whether it is within an accepted range. For example, the selection process could control the spent time for solving a specific task – it would accept a collected solution if the spent time is shorter than a predefined upper limit and longer than a predefined lower limit. Obviously, there are also other techniques available. Consider a scenario where only solutions are accepted, when the participant passes a test or control question.

3.1.1 Six Sigma

The *Six Sigma* method used in this work is inspired by the implementation of Minder and Bernstein's *CrowdLang* as presented in their paper [Minder and Bernstein, 2012] as covered in section 2.3. Originally, *Six Sigma* was implemented in the industry by Motorola in the 1980s in order to

measure the quality of their produced goods [Craig Eric Schneier et al., 1995].
To measure the quality of an answer, some measurable characteristics are needed which should be provided by the output. In this context *time* is simple to measure. When one knows how long it took to provide an answer, one can use this information to say something about the quality of this given answer. Imagine a task where workers have to describe an image with 5 different labels. In average it took, for example, 21 seconds to solve that task.
But if a specific worker solved the assignment within 3 seconds, one should be concerned about the quality of his labels. The worker probably didn't spend enough time to think about good matches – he just provided some very basic or even random labels.
However, if another worker spent for the same task for example 165 seconds, one should also be concerned about it. He provided probably a too specific solution. Instead of giving a label such as *car* or *sports car*, he had for example provided a label like *two-seated red 1998 Ferrari Maranello 550*. This is of course very accurate but it is hard to process it, since it contains multiple words for which a category mapping is difficult.

The application of the Six Sigma method includes the following steps:
Input: n answers λ from m solved tasks and a working time ω for each answer λ, to be checked using a *Six Sigma* filter.

1. Let the initial container k consist of n tuples $[\lambda, \omega]$
2. Calculate the average working time $\bar{\omega}$:

$$\bar{\omega} = \frac{1}{n} \sum_{i=1}^{n} \omega_i \tag{3.1}$$

3. Calculate the working time variance s^2 for every tuple $[\lambda, \omega]$:

$$s^2 = \frac{1}{n} \sum_{i=1}^{n} (\omega_i - \bar{\omega})^2 \tag{3.2}$$

4. Using the variance s^2, calculate the standard deviation σ as follow:

$$\sigma = \sqrt{s^2} \tag{3.3}$$

5. Using σ, calculate the *Six Sigma* specific limits:

$$\text{Upper limit } \alpha : \bar{\omega} + 3\sigma \tag{3.4}$$

$$\text{Lower limit } \beta : \bar{\omega} - 3\sigma \tag{3.5}$$

6. Check for each tuple $[\lambda, \omega]$ whether the working time ω is within the limits of *Six Sigma*: *if* $\omega \leq \alpha$ and $\geq \beta$

then add the specific tuple [λ, ω] to a container k'

7. Container k' with size $s \leq n$, consists of all tuples [λ, ω] which holds the condition of the *Six Sigma* filtering.

3.1.2 Gold Standard

The most important requirement for the implementation of a *Gold Standard* quality assurance mechanism is the ex ante availability of a list with true values for a set of *gold questions* or tasks respectively (these values could also be calculated by a crowdsourcing approach but this is not in the scope of this work). The *gold questions* will get mixed with unsolved questions and then published as a task on a crowdsourcing platform such as Amazon's Mechanical Turk. After the finished assignment, the requester will collect all answers and with the help of the *gold questions* he will be able to make an assumption about the provided quality. If the answers of the *gold questions* hold a certain level of quality, it can be assumed that the other *unknown* questions are also solved with the same precision. In contrast to the *Six Sigma* filtering approach, the *Gold Standard* method need some special task preparation:

Requirements & Preparation: Generate p gold questions in order to create m *Gold Standard* tasks. The size of p should be in the same ratio as the mixture of the task design. If for example 2 out of 6 questions in a task are gold questions, then the size of p should be $\frac{1}{3}$ of the n unsolved tasks.
Input: m tasks build out of p gold questions and n unknown questions, so every task has the same ratio of gold questions and unknown questions.

1. Calculate the quality of the solved task m_i by comparing the answers of the gold questions p_j with the true values. Use an appropriate evaluation method from chapter 4.2
2. If the quality of the *gold question* answers is equal or lower (for Pie Chart Estimation task and Data Quality task) or higher (for Rewriting task) than a predefined error rate, then accept the entire answer set of task m_i
3. If the answer set was accepted in step 2, store the new gained answers as tuples of [λ, m]
4. Repeat step 1-3 for all tasks m_i

For the Pie Chart Estimation task as well as the Data Quality task, the error rate level was set at .2, for the rewriting task. For the Rewriting task, the quality of gold question answers had to be equal or higher than .3.

3.2 Aggregation mechanisms

Since every task was solved multiple times by different workers, it is not enough to just filtering the answers by *Six Sigma* or by a *Gold Standard* approach. There could still be multiple answers for

each task which have passed the filtering control mechanism. Therefore it is necessary to combine the filtering methods with one of the following aggregation mechanisms in order to accumulate those multiple answers into a single best solution.

3.2.1 Average/Mean

Applying an average method for aggregating a set of outputs is a very simple solution. It is the calculated average of all values and it is rarely a bad choice to get a first accumulated output. This mechanisms will also be applied to the use case about *Pie Chart Estimation*, explained in more detail in section 4.1.1.

Premise: the answer is of type *number*, so the task is for example about estimating a number
Input: n answers from task m_i

1. Sum up all answer values λ from task m_i and calculate average $\bar{\lambda}$ of those values:

$$\bar{\lambda} = \frac{1}{n} \sum_{i=1}^{n} \lambda_i \tag{3.6}$$

2. Store the average value $\bar{\lambda}$ as a new tuple [m, $\bar{\lambda}$] in the container k''
3. Repeat step 1 and 2 for all tasks m_i

3.2.2 Median

The median is another mathematical value which helps to formulate a statistical statement about a sample or a distribution. It splits the higher half of it from the lower one and is more resistent against outliers. This method will also be applied to the *Pie Chart Estimation* experiment (section 4.1.1).

Premise: the answer is of type *number*, so the task is for example about estimating a number
Input: n answers from task m_i

1. Sort all n answers from task m_i in ascending ordering
2. *if n mod* = 0
   then define median by drawing the element at the position $[\frac{n}{2} + \frac{n}{2} + 1]$ in the ordering.

   if n mod $\neq$ 0
   then define median by drawing the element at the position $[\frac{n+1}{2}]$ in the ordering.
3. Store the median value $\hat{\lambda}$ as a new tuple [m, $\hat{\lambda}$] in a container k''
4. Repeat step 1 - 3 for all tasks m_i

3.2.3 Mode and Output Agreement

The term *mode* is more often used in mathematical context and the term *output agreement* is often seen as a general term, but both of them work in the same way. Popular applications of this approach are the reCAPTCHA service or the ESP Game, both invented by Luis von Ahn [Law and von Ahn, 2011]. As soon as two or more people agreed on the same description or word respectively, it is considered as a true value – in the case of the reCAPTCHA service this method helps to digitize documents such as books. This aggregating method will be applied to all three use cases (sections 4.1.1, 4.1.2 and 4.1.3), since the underlying value type does not matter.

Premise: the answer can be of any type – so the answer could also be of type *string*
Input: n answers from task m_i

1. Sort all *n* answers from task m_i in ascending ordering
2. Count for every value λ how often it occurs in the sorted list
3. *If* more than one value has the same highest occurrence, *then* take one of them by random.
4. Save the value with the most occurrence as a tuple $[\acute{\lambda}, m]$ in a container *k"*
5. Repeat step 1 - 4 for all tasks m_i

3.2.4 Voting

A voting method can provide as an output also one single solution like the previous rather mathematical methods do – but the voting method differs from them by the requirement of human interaction. This can be an advantage which also gets underlined by a statement of Sun and Dance [2012], in which they say *"...that humans are better at comparing results to pick the correct one than at producing correct results."*
As a limitation of this work and as of cost reason we draw 4 values from a distinct list and use them to generate a voting for each task m_i. Since the voting method is executed by human workers, the underlying value type is not that important for the calculation – therefore this method can also be applied to all three use cases for a subsequent evaluation (sections 4.1.1, 4.1.2 and 4.1.3).

Premise: the answer can be of any type – so the answer could also be of type *string*
Input: n answers from task m_i

1. Create a distinct list *l* of all values from task m_i
2. Draw 4 values from this distinct list *l*
   (*If* the task template requires 4 values but the distinct list does consist of less than 4 values, draw all of them and add placeholder such as null values for the remaining fields)

3. Generate a voting task with these 4 values
4. Let solve this voting task by one or more workers
5. Apply an *output agreement* on all given answers resulting from the voting task
6. Save the aggregated value from step 5 as a tuple $[\dot{\lambda}, m]$ in a container k''
7. Repeat step 1 - 6 for all tasks m_i

4

Experimental Setup and Use Cases

This chapter shall be devoted to the experimental setup and the use cases that were developed and conducted. The quality assurance mechanisms, presented in chapter 3, were applied on different use cases, which were then published to Amazon's crowdsourcing platform *Mechanical Turk*. The entire preparation of the experiment, including the collection of input data, was executed by self-written Java applications. The subsequent evaluation by reading in the collected output was also performed by Java applications. In order to evaluate the quality and performance respectively of the proposed quality assurance mechanisms, it is necessary to collect large sets of data by developing use cases (section 4.1) and their subsequent publishing to a crowd. The data sets are then run through filter methods like the *Six Sigma* method (section 3.1.1) and the *Gold Standard* approach (section 3.1.2). Eventually, the quality of the output will be calculated by appropriate evaluation metrics, which are described in section 4.2.

4.1 Use Cases

In order to evaluate the presented quality assurance mechanisms, it is indispensable to run field experiments. Thus, the following section covers the use cases, which were developed for the experimental phase. These use cases contain three different types of crowdsourcing tasks, covering *assignments* which are often seen on platforms like Mechanical Turk and includes estimating numbers, verify data sets and search for additional information as well as improving existing data. In subsection 4.1.1, 4.1.2 and 4.1.3 the three use cases are presented.

4.1.1 Pie Chart Estimation Task

The *Pie Chart Estimation* task is about estimating a number – workers have to estimate the size of a highlighted pie chart slice, displayed in a diagram and must provide their answers as a

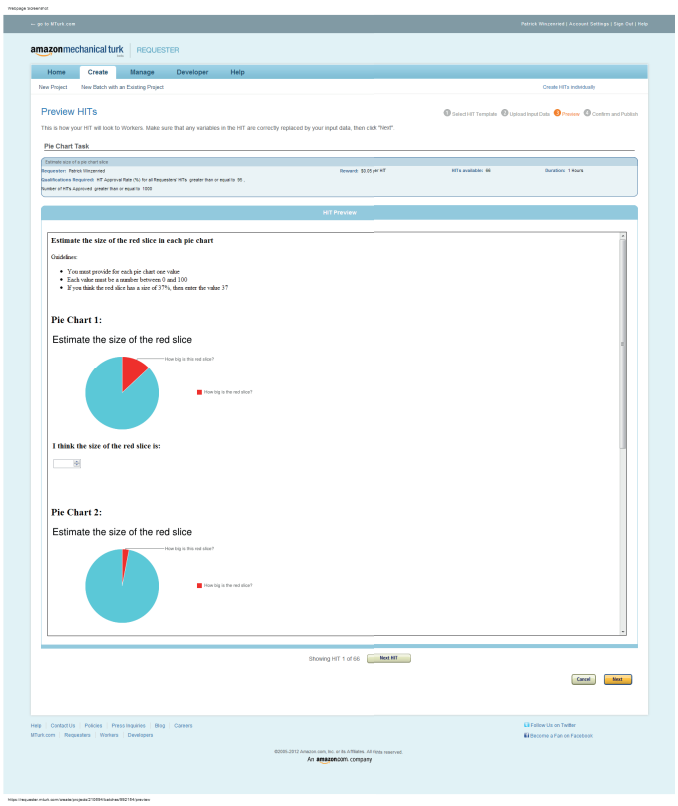


Figure 4.1: Preview of Pie Chart HIT, showing two of three Pie Charts

percentage between 0 and 100. They do not get any additional information about the task beside of a brief task description to support the input procedure. The task does also not require any know-how or knowledge that should have been trained in prior, so it should be manageable for every crowdsource worker to solve such an assignment immediately after reading the task description. This kind of task may not be directly mapped to a business issue but in terms of research it shows how well a crowd works. The wisdom of the crowd provides estimators which are surprisingly good – Francis Galton found that not one of 787 different persons could guess the exact weight of a living ox at a weight-judging competition but the average of all answers – Galton called it *vox populi* [Galton, 1907] – was amazingly accurate. The ox had a weight of 1198 pounds, whereas the average of the crowd was just one pound off, namely 1197 pounds. The average of all given answer is a simple but effective way of guessing it. This use case shall therefore help to understand how accurate workers can estimate a size, in this case a slice of a pie chart diagram and whether a subsequent filtering method does improve the aggregated output. Consider figure 4.4 in order to see how the total process of this task looks like.
Figure 4.1, a preview of a *Pie Chart Estimation* task, shows how this use case looks like in an actual Mechanical Turk assignment.

4.1.2 Data Quality Task

Consider a situation where someone, for example a company, has a customer list with hundreds of outdated entries from earlier days. It would be time consuming and boring for a clerk to update this list manually in order to get the latest homepage URLs, e-mail addresses or verified business addresses. Wouldn't it be a huge relief when such a task could be outsourced?
This use case simulates exactly such a situation. A set of business information such as a company name, a phone number and a postal address is presented to a worker which then has to look for the corresponding business homepage. The input for this task was manually picked from Canada's Yellow Page website (http://www.yellowpages.ca) in order to make sure that only useful data sets without any ambiguousness were collected. Obviously, the purpose of a use case like this is to find out whether the outsourcing of such data quality task and the following filtering of the work products can be performed by a certain quality level. Figure 4.5 shows how the entire process of this tasks looks like.
The presentation of business information sets is relatively simple and a short task description is enough to explain what exactly is needed from a worker who chooses to solve the task successfully. Following figure 4.2 shows how a *Data Quality* task on Mechanical Turk looked like which was created for the experiment. The composition is similar to the *Pie Chart Estimation* task, however this task contained 5 data sets per HIT. As an additional constraint workers were asked to provide only top level domains, no sub domains or domains with suffixes like */index.html*.

4.1.3 Rewriting Task

High quality translations by human experts are very expensive and time consuming. Machine translation services are a cheap and fast alternative but the output quality is quite bad – if a professional translation is needed, for example for a multilingual company website, there is almost no option other than hiring a translation agency. But with an appropriate task design, it could be possible to outsource such a task to the crowd – a worker would propably be willing to translate a phrase or a paragraph for a small compensation. Such an approach would be much cheaper than a professional translation service. In order to create such an experiment, this use case is about improving paragraphs which are written in English. The input originates from *Project Syndicate* (http://www.project-syndicate.org), a website that publishes collected articles about economy and politics which they consider as important. They translate many of them into different languages such as German, French, English and Spanish. The entire process flow for this task is presented in figure 4.6.
100 paragraphs which are written in German and their existing counterparts in English were manually collected, as well as corresponding leading and following paragraphs in order to provide some context. The German paragraphs were then translated into English by a translation service. As already pointed out the machine translated text was far from perfect – so the worker had to rewrite these paragraphs in order to improve them. Such a machine translated paragraph is

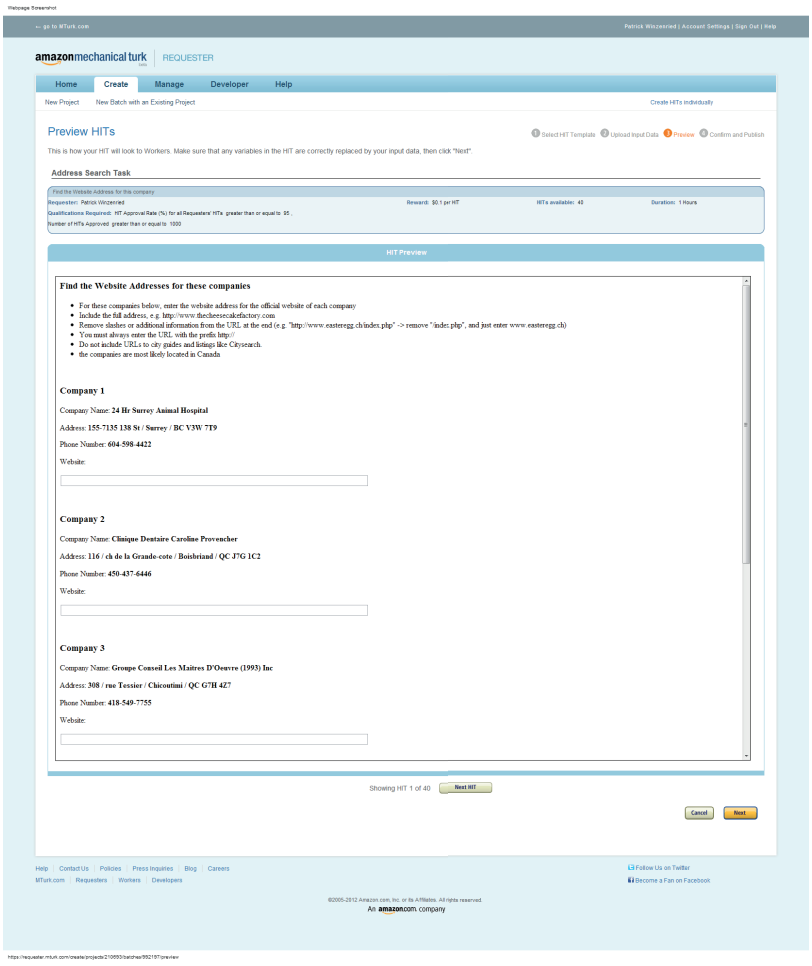


Figure 4.2: Preview of a Data Quality Task, showing three of five business address sets

presented in the following figure 4.3.

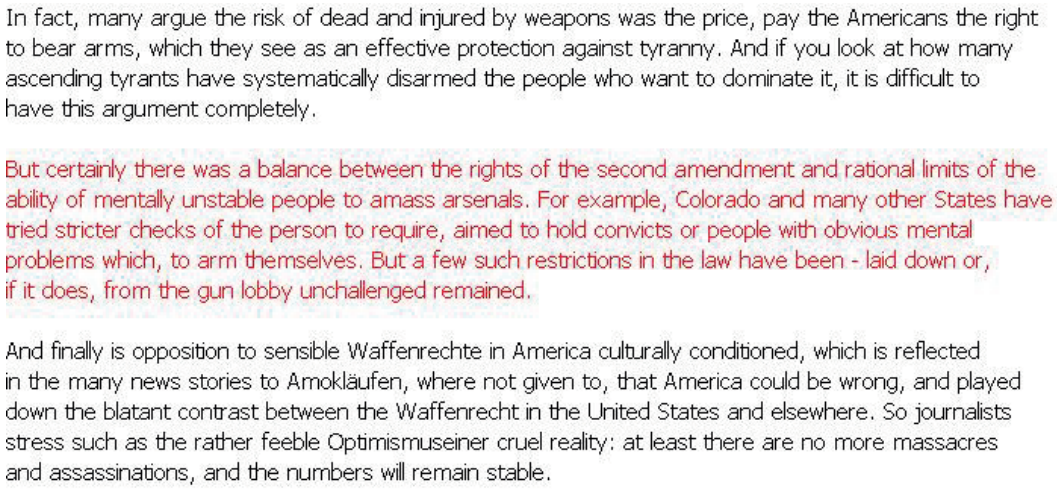


Figure 4.3: Example of a Rewriting Task image, the worker has to focus on the red highlighted text

4.2 Evaluation Metrics

In order to formulate a statement about performance characteristics like reliability and efficiency, it is inevitable to compare the output of a quality assurance mechanism against a set of true values. On the basis of such an evaluation it is then possible to discover statistical relations and leads eventually to a final statement in terms of quality, costs and time.

As already indicated, the evaluation of a method such as a quality assurance mechanism is only possible if a true or target value is available for a given task.
One has then the ability to measure the performance with mathematical or statistical tools. If large deviations between the output and the target are discovered, it may be necessary to adjust the mechanism in order to refine it. But it could, of course, also reveal major disadvantages, which probably would lead to the application of another metric or even the engineering of a new method.
In summary, it is necessary to prepare a set of control tasks as part of the evaluation process – with those control tasks, for which the true values are known, it will be then possible to formulate a quality statement, for example, like "The quality assurance mechanism XY provides in 89% of the opportunities an acceptable output below the error rate of 0.05 – by costs of $ 0.05 per task and an average working time of 24.3 seconds.".

The following subsections will explain in details how the individual evaluation metrics work and how they are applied.

4.2.1 Mean Squared Error

The mean squared error, also called root mean error, is a popular and simple way of weighting the difference between an estimator and a true value. The error indicates how strong these two values differs from each others. An error rate of zero would be interpreted as a perfect estimation of the true value, so the estimation would be equal the true value. Thus, the overall goal is to get as close as possible to an error rate of zero.
The mean squared error is applied on a set of estimations and true value and then calculates the average error rate of this set. Since the difference between the estimator and the true value is squared, bigger differences are weighted more than smaller ones – and as a side effect the error rate is always positive. This evaluation method works when the underlying values are of any kind of numbers. The unit of measurement does not matter on this occasion – the mathematician Norman Levinson used the mean squared error, for example, to analyze the occurrence of *noise*[1] when transmitting information by mechanical or electrical means [Nohel et al., 2012]. To be exact, he measured the difference between a sent signal and a received message which could contain *noise*.

This evaluation method is therefore an ideal choice for the use case about *Pie Chart Estimation* in section 4.1.1, where the estimation as well as the true values are also of the type *numbers*. The application of the mean squared error is straight forward since it does not need any kind of value transformation or mapping to a scale.
What follows now is the mathematical expression of the error calculation as well as additional details about the application of the mean squared error on the *Pie Chart Estimation* use case.

$$\text{Mean Squared Error} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \tag{4.1}$$

For illustration purposes, the following table shall be considered. It shows three tuples of true values and estimations, for example, from a *Pie Chart Estimation* task. The last column shows the squared difference of each tuple.

According to the expression given above, the value of the mean squared error rate would be the total of column $(\hat{Y} - Y)^2$ divided by the number of rows – in this case 3, the mean squared error is then *0.00177*. This is quite near to zero and could indicate that the used estimation mechanism is probably a good choice – but this can only be formulated when it is compared to other kinds of prediction methods or compared against a base line – and it should, in order to verify the robustness, also be run with sets containing significantly more than 3 tuples. Because only with

[1] In that connection, *noise* is a kind of interference that occurs in electrical circuits

True value	Estimated	$\hat{Y}$ - Y	$(\hat{Y}$ - Y$)^2$
0.89	0.90	-0.01	0.0001
0.56	0.62	-0.06	0.0036
0.28	0.24	0.04	0.0016

Table 4.1: Small example to demonstrate mean squared error calculation.

an additional row, for example, a true value of *0.10* and an estimation of *0.85* (the participant may probably have misunderstood the task), the mean squared error would be *0.14195* – which is much worse than the first error rate.

4.2.2 Levenshtein Distance

For the use case about *Data Quality* (section 4.1.2), workers were asked to find URLs for given business addresses. As part of the evaluation process, these URLs have to be compared against the true domain names. However, a simple string matching would not be sufficient because this kind of test would only tell whether a string is a perfect match of the true value – so it would only give back true or false statements.
Thus, it is inevitable to apply another evaluation metric which works in a more sensitive way. The evaluation metric should provide detailed information about the comparison of two strings – for example, consider the URL *http://www.uzh.ch* as a true value and the URL *http://www.uzh.ch/index.html* as an answer from a worker. As explained in the use case about *Data Quality* (section 4.2), the worker should have provided only the top level domain without any additional parts such as *index.html*. An appropriate evaluation metric would nonetheless show that these two URLs are not entirely different by calculating an error rate for it – similar to the mean squared error explained in section 4.2.1.
Such kind of information provides the method developed by Vladimir Levenshtein in 1965 [Levenshtein, 1966], called *Levenshtein distance* – which is nowadays often used for spell checkers or prediction functions, for example, Google's real-time suggestions when using the search engine.
Levenshtein analyzed the smallest effort (or costs) which is needed to get from one string to another and formulated a mathematical formula to calculate the distance between two strings. This can be best understood by illustrating an example: Consider the words *couch* and *roach* – each deletion, insertion or substitution of a character is considered as one unit of the *Levenshtein distance*.

1. couch $\rightarrow$ rouch (substitution of *c* for *r*)
2. rouch $\rightarrow$ roach (substition of *u* for *a*)

The *Levensthein distance* in this case is 2.

If two words are totally different from each other, the *Levenshtein distance* would be equal the size of the largest word; consider as an example the words *house* and *cat*:

1. house → couse (substitution of *h* for *c*)
2. couse → cause (substition of *o* for *a*)
3. cause → catse (substition of *u* for *t*)
4. catse → cats (deletion of *e*)
5. cats → cat (deletion of *s*)

The *Levenshtein distance* is in this case 5, this is equal to the number of characters in the word *house*.

To get the same kind of error information as the mean squared error provides, a simple division of the *Levenshtein distance* by the size of the largest word is necessary. The error rate for the first example would then be $\frac{2}{5} = 0.2$ and for the second example it would be $\frac{5}{5} = 1$, where a value of 1 indicates that the two strings are totally different and a value of zero would confirm that both strings are identical. With this evaluation metric, the URLs *http://www.uzh.ch* and *http://www.uzh.ch/index.html* mentioned in the beginning of this section would receive an error rate of $\sim$0.39.

4.2.3 Meteor

When it comes to the evaluation of whole phrases instead of simple strings or single words, the *Levenshtein distance* is not sufficient anymore. Consider, for example, the output of a translation task, which has to be compared against a reference translation. The phrases *Peter visited his grandmother when it was a sunny day* and *Peter visited his grandmother on a sunny day* do not differ from each other very much. The *Levenshtein distance* would, however, only consider the exact ordering of the characters and probably calculate a rather bad error rate.
A better evalution metric should weight the choice of words (compared against a reference sentence) as well as their ordering. A method which takes these characterstics into account is called *Meteor*.

Meteor is an evaluation metric method for machine translations – demonstrations have proved that it performs in a same way as human experts would evaluate translations – but of course faster and cheaper [Lavie and Agarwal, 2007].
Meteor computes a meteor-score based on a word-to-word alignment between a translation and a given reference translation.
In a first step it calculates a parameterized harmon mean of a precision value $P = m/t$ and a recall

value *R* = *m/r*, where *m* are the two phrases, *t* is the total number of unigrams in the translation and *r* the number of unigrams in the reference sentence [Lavie and Agarwal, 2007]:

$$F_{mean} = \frac{P \cdot R}{\alpha \cdot P + (1 - \alpha) \cdot R} \tag{4.2}$$

In a second step the ordering of the word is taken into account. *Meteor* computes therefore a *penalty* value by detecting *chunks* of matching word orderings, which then get divided by the number of matchings. The final score computation is provided in the following expression:

$$score = (1 - Penalty) \cdot F_{mean} \tag{4.3}$$

The *Meteor* implementation used in this work is provided by the Carnegie Mellon University as a Java library (version 1.4) [Denkowski and Lavie, 2011].

4.2.4 Costs

The calculation of the costs is relatively simple and straight forward. The costs for a single task is calculated on the basis of the fee charged by Amazon's Mechanical Turk platform.
The actual setup of the experimental tasks is explained in the following section 4.3 – the calculation example provided here shows how the costs for a single task of the *Pie Chart Estimation* is calculated and is a detailed excerpt of the section 4.3.2.
Amazon charged a total of $36.3 for 660 assignments – which includes a 10% fee. The 660 assignments are comprising of 66 HITs (Human Intelligence Task) each solved by 10 different workers. Each HIT was a set of 3 estimation tasks. Thus, the costs for a single task are $0.01833:
$\frac{36.3}{660}$ = 0.55 (Costs for a single HIT) $\rightarrow$ $\frac{0.55}{3}$ = $0.0183.

4.2.5 Time

The capture of the time is by far the easiest challenge. Mechanical Turk measures how long it took for a worker to solve a task. It provides then this information along with the output and maps it also to each individual worker ID. The working time provided in section 5.2 is an average working time of all provided assignment solutions. In the case of the *Pie Chart Estimation* tasks, this would be a total sum of all 660 captured working times and then divided by the total number of assignments. On average, it took 16 seconds to solve a pie chart estimation task for example.

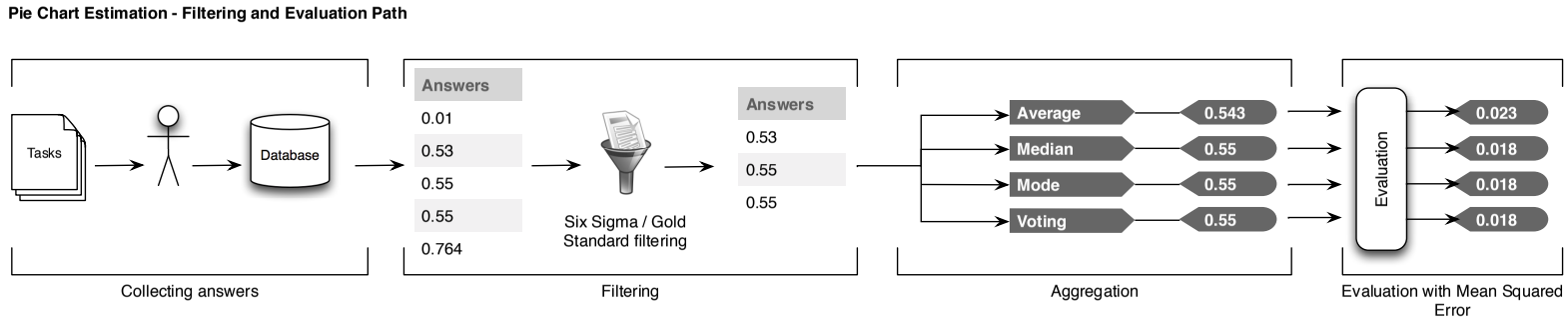


Figure 4.4: Process of filtering, aggregating and evaluating the Pie Chart Estimation experiment - with all 4 aggregating methods (simplified data sets)

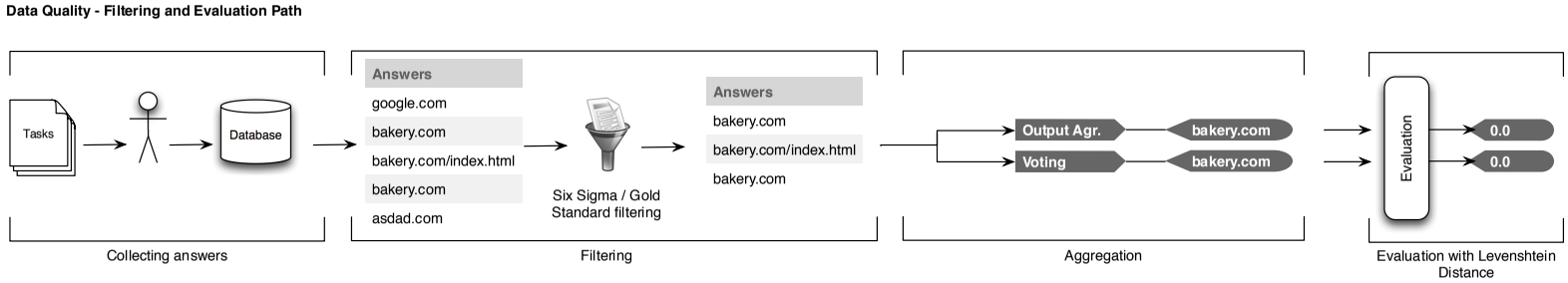


Figure 4.5: Process of filtering, aggregating and evaluating the Data Quality experiment - output agreement and voting aggregating methods (simplified data sets)

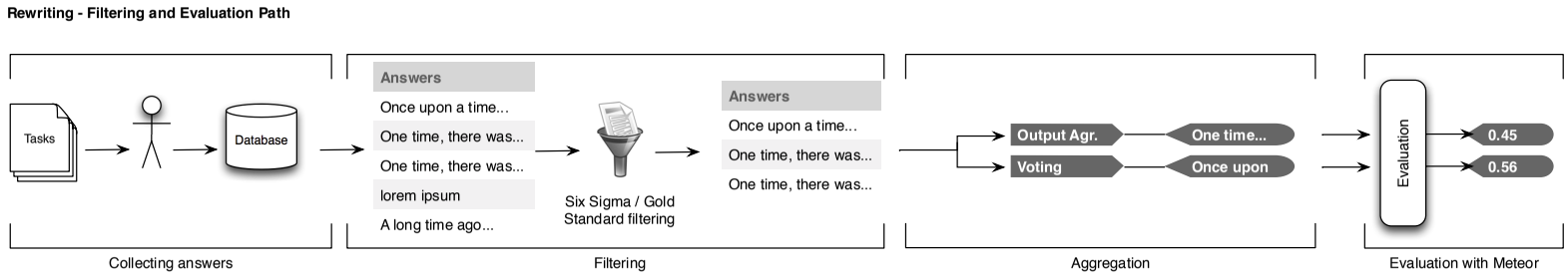


Figure 4.6: Process of filtering, aggregating and evaluating the Rewriting experiment - with output agreement and voting aggregating methods (simplified data sets)

4.3 Experiments

The actual design of the tasks and the execution of the experiments were done on Mechanical Turks platform – it provides an HTML environment with a WYSIWYG editor, which enables a requester to create a master template – one can insert input fields and set HTML constraints like on a normal HTML website – so it is possible, for example, to accept only numbers between 1 and 100 or accept only URLs as an input. The template consists also of text such as the detailed task description and of placeholders such as $\{input\_1\}$, $\{input\_2\}$ and $\{input\_3\}$ which will be replaced by the time of the task publishing with the actual input provided by an uploaded CSV input sheet. The input can be of any type – it is possible to replace the gaps with paragraphs or even URLs to show, for example, an image. In the same instance, the requester also defines how much he wants to pay for a task – which is usually something between $0.01 and $1. He can also set additional constraints as, for example, that only workers are allowed to solve the assignments, who already did more than 1000 assignments with an approval rate of 95%. Mechanical Turk calculates then the total costs and adds a 10% fee for their service – detailed calculation are also shown in the following sections.
It is important to note that each HIT was solved by 10 different workers which led to a huge amount of data, as described in the following section 4.3.1 - but this is also important because it allows to create a simulation where it is possible to see how the level of quality changes with increasing number of answers. A statement like this could then be possible: *"A Six Sigma filtering in combination with an median aggregating mechanism provides a higher level of quality as soon as 6 answers are available - with less than 6 answers, a combination with an average aggregation brings the best performance."*
In order to achieve a smoothing of outliners, the procedure of picking values randomly to create the different levels of answers was executed 200 times - then the average of these 200 draws was calculated.

After the tasks are published on the platform and workers solved all of them, Mechanical Turk arranged the answers in an output CSV file, which contains the original input (with the unique naming for each input), information about the task such as IDs, date and time as well as information about the workers, like for example, a unique worker ID and the working time spent for each task. This output has then to find its way back into the SQL databases so it can be used for further computations such as quality metrics (section 4.2). This was done by a Java application which reads in the CSV output and matches every answer, based on the unique input naming, to the original input stored in SQL databases. Eventually, the database holds for each input the true value, all information about the HIT and the worker and of course the collected answer. The whole procedure of collecting the input (1), preparing it according to Mechanical Turks requirements (2), creating the templates (3), publishing the tasks to the crowd (4), collecting the answers (5) and reading them back into the SQL database (6) for further use is shown as an

illustration in the following figure 4.7:

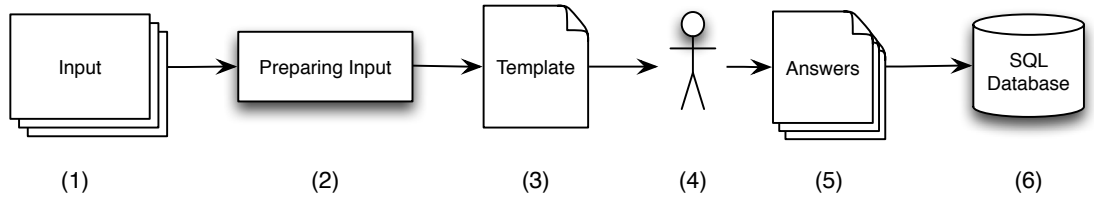


Figure 4.7: Procedure of collecting and processing input, publishing it to the crowd and collect and store the answers into a database

4.3.1 Descriptive Statistics

This subsection summaries some statistical information about the experiments like the total number of HITs, how much assignments were performed for each of them and how many answers were collected.

The input for the *Pie Chart Estimation* task originates from an application, which simply created 198 pie chart diagrams by choosing random percentage values. The actual task was then designed in such a way that 3 diagrams were displayed as a group in one HIT, as shown in figure 4.1. Thus, the 198 diagrams generated in prior resulted in 66 HITs and each such HIT had to be solved by 10 different workers. This led to 660 assignments, whereas each of them collected 3 answers what makes 1980 answers in total, eventually.
The reward for solving a single assignment was set at $0.05 – consequently, this task had cost $33, adding a 10% fee for Mechanical Turk's service, this led to a total of $36.30.

After collecting manually appropriate paragraphs for the *Data Quality* task the input database hold 200 business entries, eventually. A Java tool prepared then the CSV input file for the task publishing in such a way that 40 HITs were created, each showing 5 business sets. Similar to the *Pie Chart Estimation* task each HIT was solved by 10 different workers – by a number of 40 HITs this makes a total of 400 assignments with a total output of 2000 URLs as answers. The payment for each assignment was set at $0.1, so the entire task costs were about $44, including Mechanical Turk's 10% fee.

The *Rewriting* task was set up as an experiment with 50 HITs, each contained two rewriting tasks. Again, each task was solved by 10 different workers which got a compensation of $0.10 per HIT. Consequently, Amazon charged $55 for this experiment.

The following table provides an overview of the input, the tasks per HIT, how many assignments were done per HIT and to what amount of output this led for each of the three experiments. A cost overview is also included by the last row.

	Pie Chart Estimation	Data Quality	Rewriting
Input	198	200	100
Sub tasks per HIT	3	5	2
Total HITs	66	40	50
Assignments per HIT	10	10	10
Total assignments	660	400	500
Total data records	1980	2000	1000
Costs per assignment ($)	0.05	0.10	0.10
Total costs incl. fee ($)	36.30	44.00	55.00

Table 4.2: Overview of experimental setup: number of inputs, sub tasks per HIT, total HITs, assignments, costs per assignment and total costs.

4.3.2 Tool Description

This section highlights the used tools and developed solutions in order to create, run, collect and evaluate the experiments. The entire experimental work was supported by self-written Java applications and all data sets were stored in PostgreSQL databases, one for each experiment.
Since Mechanical Turk requires comma separated value lists for the batch processing, it was necessary to implement methods which could create such CSV lists out of the data stored in the databases.

Unfortunately, Mechanical Turk does not allow hidden columns in their CSV input sheets – so it is not possible to add some kind of input identification in order to match the afterwards collected worker output easily with the data in the SQL databases. Thus, it was necessary to add some sort of identification directly into the input. In the case of the *Pie Chart Estimation* this was however quite simple – since the input was a URL, which directs to an pie chart image – it was possible to give each image an unique name, assigned by the Java application which prepared the CSV input files. The same was possible for the *Rewriting* task but not for the *Data Quality* task – but since every business record consists of a unique phone number, this was used as an identifier. The next paragraphs explains how the input was actually prepared for the experiments.

Pie Chart Estimation Task

The input was created by a self-written Java application which uses a functionality provided Googles Web ToolKit (source) in order to create diagrams. A diagram can be created and displayed online by using a URL[2] which contains all the information to create an individual pie chart.

[2] `http://chart.apis.google.com/chart?cht=p\&chs=500x250\&chdl=How+big+is+the+red+slice\%3F\&chl=How+big+is+this+red+slice\%3F\&chco=FF0000|00FFFF|00FF00\&chp=4.712325\&chtt=Estimate+the+size+of+the+red+slice\&chts=000000,24\&chd=t:+redSlice+blueSlice`

redSlice and *blueSlice* (at the very end of the URL) must be replaced by two numbers, which have to result together in a total of 100. This results in an image like shown in figure 4.8, where the highlighted red slice has a size of 56% and corresponds to the placeholder *redSlice*. It should be noted that the entire image legend, which is normally provided by the ToolKit, is removed in order to avoid any information disclosure to the worker, so he could only guess the size of the highlighted slice by looking at the image.

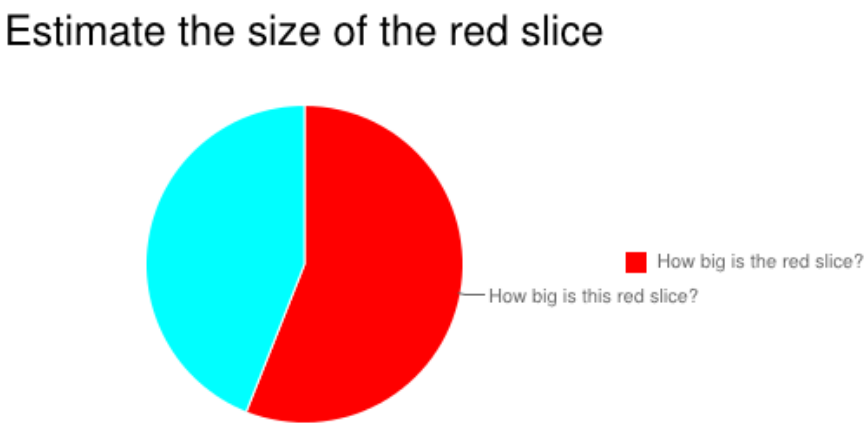


Figure 4.8: Example of a PieChart generated by Google's Web ToolKit. URL with a red slice size of 56%

Data Quality Task

The input for the *Data Quality* task was mainly created by hand. However, some support was provided by Java application in order to generate the CSV batch file. Eventually, each task contained 5 sets of business information. As an input constraint, each of the 5 input fields only accepts answer in the format of a URL – so the prefix *http://* was necessary – an excerpt of the corresponding HTML code is shown below – the placeholders as well as the additional type conditions are highlighted.

```
<h3><b>Company 1</b></h3>
<p>Company Name: <b> ${name1} </b></p>
<p>Address: <b> ${address1} </b></p>
<p>Phone Number: <b> ${phone1 } </b></p>
<p>Website:</p>
<p><input <b> type ="url" size="100" id="URL1" name="URL1" required="required" /></p>
```

Listing 4.1: HTML snippet of the Data Quality Task

Rewriting Task

The preparation for the rewriting task was relatively easy but time consuming. Although, all 100 paragraphs, which were needed as input for the experiment, were manually collected, the actual translation part was thanks to the open API of Microsoft Bing's translation service (http://www.bing.com/translator) executed automatically by a Java application. The generating of the batch file was done in the same way as for the other two tasks.
After each of the three parts (leading, main and following part) of a rewriting was translated, a Java application joined the parts and transformed them into JPG image file (see figure 4.3 on page 23), so a worker cannot simply copy the text and use some translation or word processing tools what makes cheating more difficult.
Since the task was also time consuming to solve and required specific knowledge about the English language, each HIT contained only two rewriting tasks. Thus, the feeding of the answers and the matching with the original input was easier compared to the other two tasks since each line of the output CSV file contained just one answer instead of three or five.

5

Results

After the presentation of the quality assurance mechanisms and the detailed introduction of the evaluation metrics, this chapter will bring everything together and shows the results of the evaluation in detail.

The focus is on the research question whether the Six Sigma filtering method outperforms the Gold Standard technique in terms of quality, time and costs.
The remainder of this chapter is structured as follows. Section 5.1 shows the evaluation for all three experiments from the quality point of view, it highlights the aggregate mechanisms which perform better than the others and shows which filtering method provides almost in every situation the best output – all of them evaluated either by *Mann-Whitney U tests* or *univariate Kruskal-Wallis ANOVA tests*. And section 5.2 will present the evaluation of the work time and the costs in order to understand which approach is faster and cheaper.

5.1 Quality

This section shall cover all aspects of the quality by providing overviews of the overall quality and how it behaves with increasing numbers of answers as well as detailed group and pair comparisons of the different experiments and filtering-and-aggregation-mechanism combinations.
Due to the limited scope of this work, all group and pair comparisons of the different combinations are done at the level of the maximum of 10 answers per task as the experiments were originally conducted.
All findings of the evaluation will be summarized in a global overview in table 6.1 on page 56.

5.1.1 Pie Chart Estimation

The first interesting thing which can be explored by using the collected data sets is to see whether it makes any difference how many workers were asked to solve a specific task. The following figure 5.1 shows that there are indeed differences.

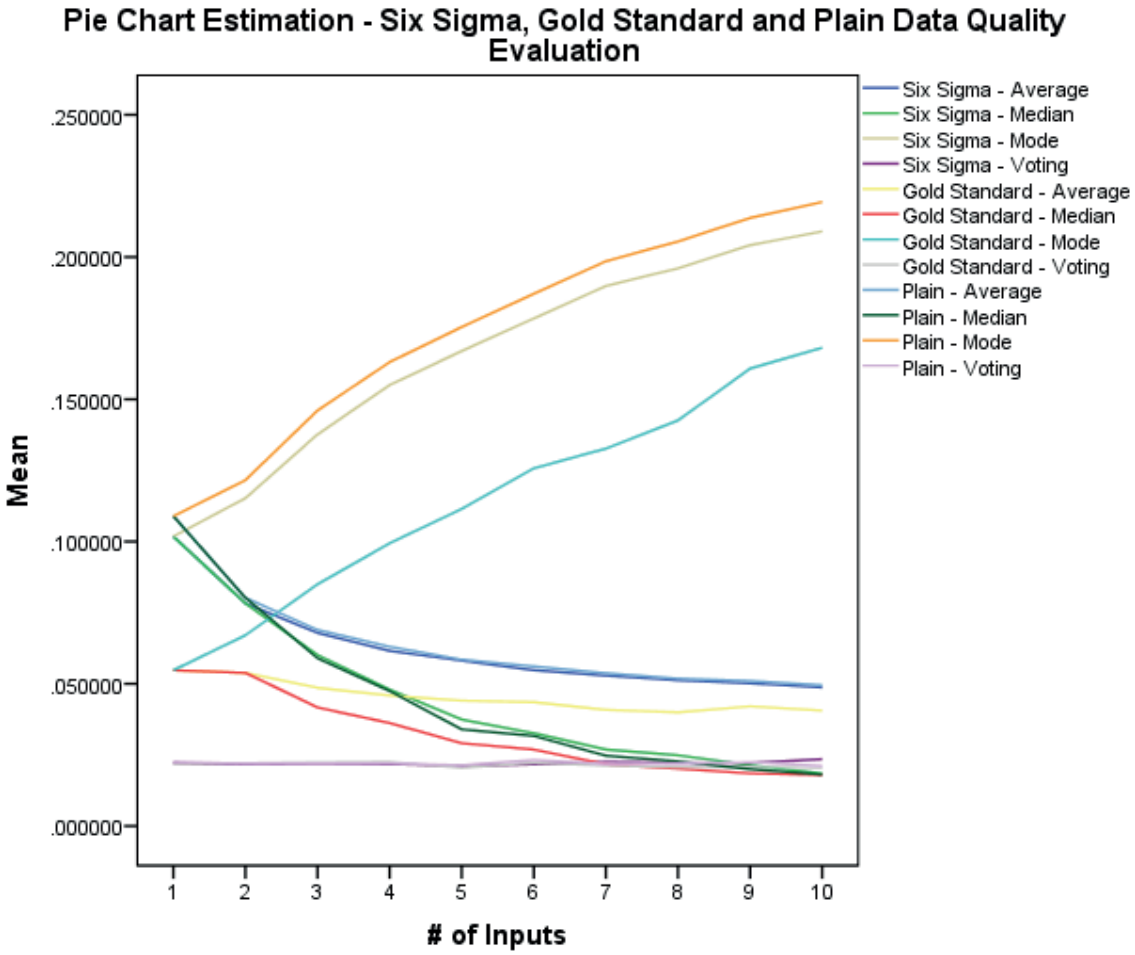


Figure 5.1: Pie Chart Estimation - Changes in quality with increasing numbers of answers. Quality is calculated by Mean Squared Error (section 4.2.1), where a value of 0 is perfect and 1 is bad.

When comparing a single input against a set of 10 inputs, then the *Six Sigma - average* combination has improved by 52% and eventually has a mean squared error of .0487 (*sd* = .0065). The *Six Sigma - median* approach has a mean squared error of .01846 (*sd* = .00995) and an improvement of 82% (comparing a single answer with a set of 10 answers). The *Six Sigma - mode* combination changed for the worse by over 105% when comparing a single input with a maximum of 10 inputs and has eventually a mean squared error of .2090 (*sd* = .0157). The *Six Sigma - voting* combination faces an improvement of 5% and has a mean squared error of .0205 (*sd* = .0135).
The *Gold Standard - average* approach improves by 26% and has a standard mean error of .0406 (*sd* = .007) at a basis of 10 inputs. The *Gold Standard - median* method has a mean squared error of .0180 (*sd* = .0096) and has been improved by over 67%. The *Gold Standard - mode* combination deteriorated by over 207% with a mean squared error of .1681 (*sd* =.0156) and the *Gold Standard - voting* technique faces a moderate quality improvement of 6% with a mean squared error of .0205 (*sd* = .008).

At a first glance it could be surprising that the *mode* aggregation method provides decreasing quality output with increasing input, independently of the combined filtering method. But this is relatively easy explained: consider a pie chart diagram where the red slice has a true size value of 24%. This could look for someone like a quarter – so if one asks 10 different people, the chance that some of them answer that the size is 25% wouldn't be very low. So the mode value (as explained in section 3.2.3) would be 25% whereas the average and median value would probably something under 25% and therefore closer to the true value of 24%. The other findings of the graphic are consistent with the expectation that the more inputs the better the output would be. For the other combinations such as *Six Sigma* with the *average* and *median* method as well as the *Gold Standard* filtering method with the same two aggregation mechanisms, it shows that the *Mean Squared Error* drops already when more than 2 inputs are considered.
Especially the *median* aggregation method works quite well with a basis of multiple answers – this is because the answers are distributed normally around the true value and thus, outliners are smoothed with additional input.

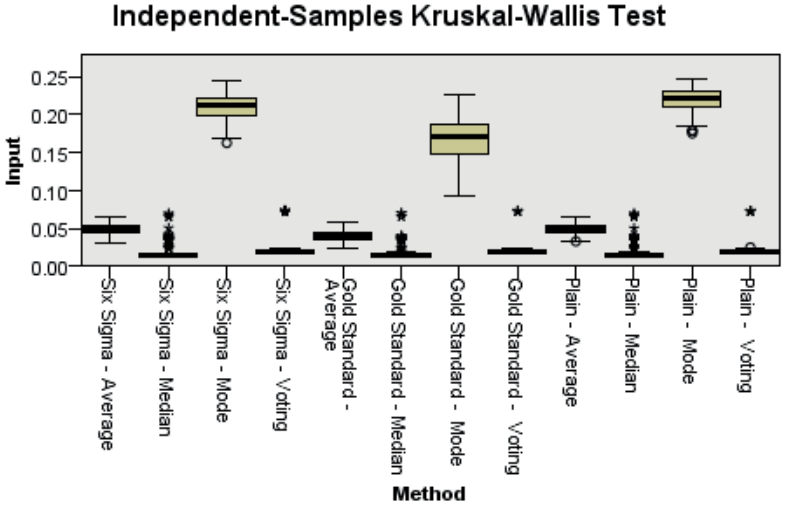
The two-sided and univariate Kruskal-Wallis ANOVA test in figure 5.2 shows that the 12 possible filtering-aggregation methods differ among each other significantly. A first superficial analysis of this graphic illustrates that the *mode* aggregation method performs worse than the other approaches. Additionally, figure 5.3 reveals that the *average*, *median* and *voting* approaches provide a robust quality independently of the combined filtering methods and it seems that either the *median* or *voting* approach will provide the most accurate output. What follows now is a detailed pair comparison between the different filtering-and-aggregating combinations, starting with a comparison of the *average* aggregation method, then the same for the other 3 methods *median*, *mode* and *voting*.

The statistical evaluation shows that the *Gold Standard* filtering in combination with the *average* aggregation method provides the best output compared to the *Six Sigma* filtering method and the *plain data* as a base line – and this even with a high significance.

The Kruskal-Wallis test (figure 5.4) for the *median* combination does not reveal at the first glance which data set provides the best quality, therefore the pairs had to be evaluated with the *Mann-Whitney U test* in addition. The results of this evaluation are very interesting, visualized in figures 5.4, 5.5, 5.6 and 5.7.

The figures on page 40 show that a simple plain data set combined with a *median* aggregation method outperforms the filtering methods easily. The significance difference was revealed by a pairwise Whitney-Mann U test with a p-value of .05.
This discovery might not be that unexpected when the idea of the median is taken into consideration: It is the value that separates a set of numbers into a lower and upper half and it is due to this fact very robust against outliners.

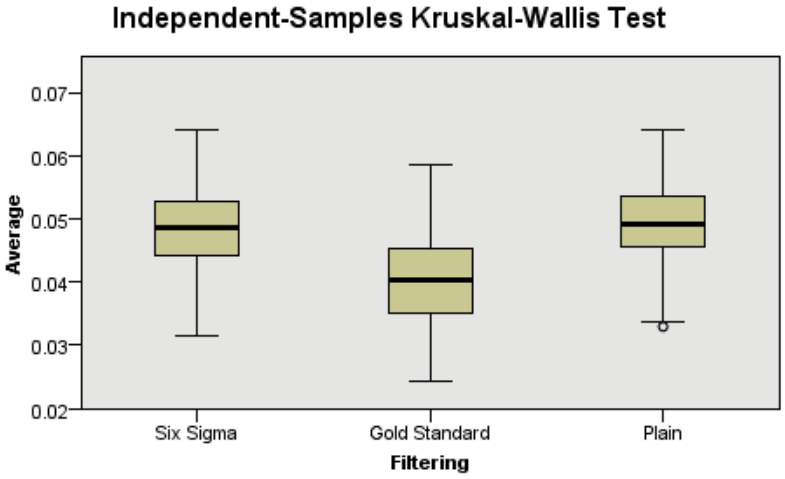
A deeper evaluation of the *mode* aggregation mechanism follows, although it shows a rather bad quality assurance according to the global evaluation in figure 5.2. The evaluation shows however, that the *Gold Standard* filtering outperforms the other two data sets again significantly, whereas the p-value was .05.



Total N	2'400
Test Statistic	2'080.032
Degrees of Freedom	11
Asymptotic Sig. (2-sided test)	.000

1. The test statistic is adjusted for ties.

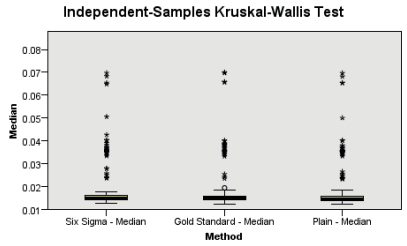
Figure 5.2: Statistical evaluation of all filtering-aggregation combinations by applying a Kruskal-Wallis ANOVA test. It shows that they differ from each other significantly with a p-value of .01.



Total N	600
Test Statistic	152.994
Degrees of Freedom	2
Asymptotic Sig. (2-sided test)	.000

1. The test statistic is adjusted for ties.

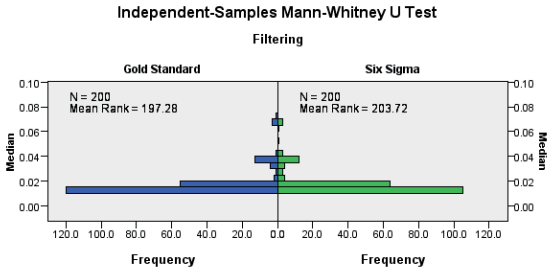
Figure 5.3: Statistical evaluation of the average aggregation method (see section 3.2.1) by applying a Kruskal-Wallis ANOVA test. The *Gold Standard-average* combination outperforms the other two combinations significantly with a p-value of .01.



Total N	600
Test Statistic	8.262
Degrees of Freedom	2
Asymptotic Sig. (2-sided test)	.016

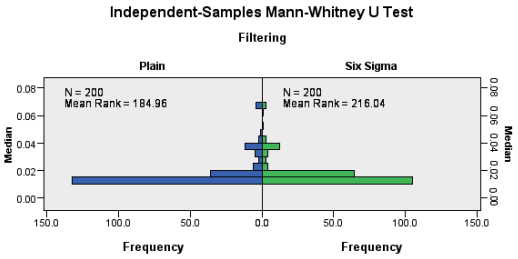
1. The test statistic is adjusted for ties.

Figure 5.4: The Kruskal-Wallis ANOVA test shows a significant difference between the three median data sets.



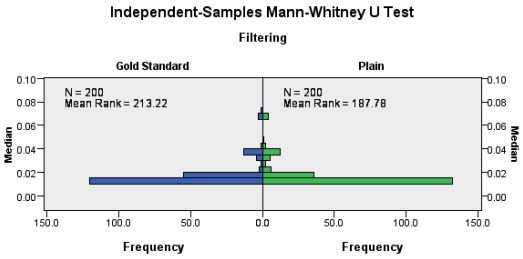
Total N	400
Mann-Whitney U	19'355.000
Wilcoxon W	39'455.000
Test Statistic	19'355.000
Standard Error	1'156.143
Standardized Test Statistic	-.558
Asymptotic Sig. (2-sided test)	.577

Figure 5.5: No significant difference between the Gold Standard and Six Sigma median data sets.



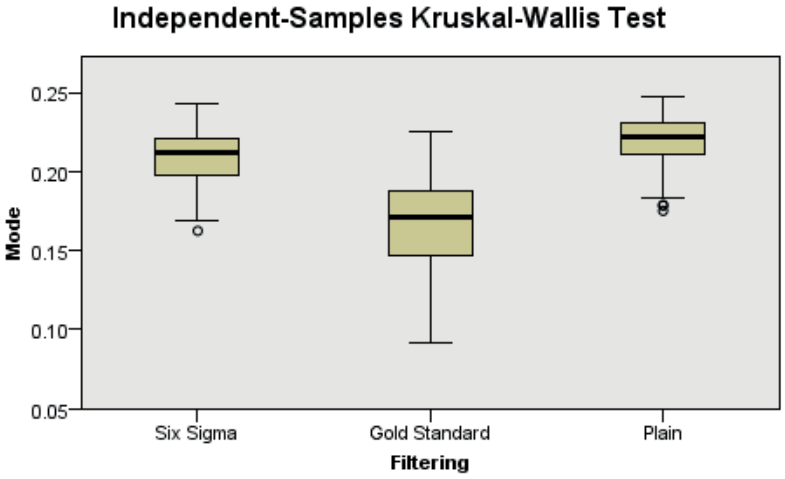
Total N	400
Mann-Whitney U	16'891.500
Wilcoxon W	36'991.500
Test Statistic	16'891.500
Standard Error	1'156.143
Standardized Test Statistic	-2.689
Asymptotic Sig. (2-sided test)	.007

Figure 5.6: The Mann-Whitney-U test shows that the median aggregation mechanism applied on the plain data set outperforms the Six Sigma method significantly.



Total N	400
Mann-Whitney U	17'455.500
Wilcoxon W	37'555.500
Test Statistic	17'455.500
Standard Error	1'156.143
Standardized Test Statistic	-2.201
Asymptotic Sig. (2-sided test)	.028

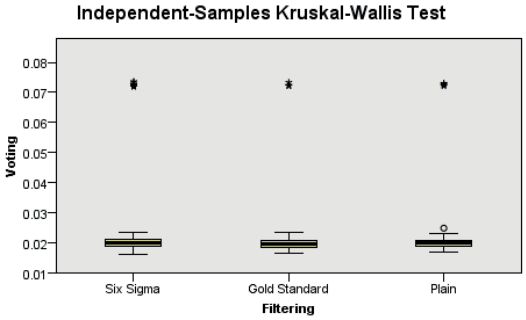
Figure 5.7: The Gold Standard filtering gets outperformed by the median application on the plain data set - the Mann-Whitney U test shows a significant difference.



Total N	600
Test Statistic	328.334
Degrees of Freedom	2
Asymptotic Sig. (2-sided test)	.000

1. The test statistic is adjusted for ties.

Figure 5.8: Statistical evaluation of the mode aggregation method by applying a Kruskal-Wallis ANOVA test. The *Gold Standard-mode* combination outperforms the other two combinations significantly with a p-value of .01.

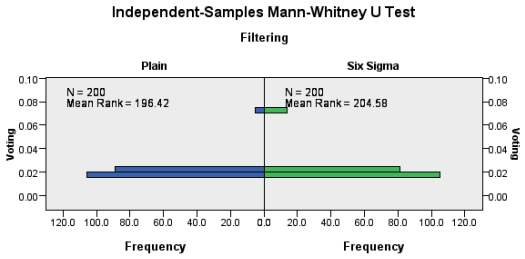


Total N	600
Test Statistic	11.412
Degrees of Freedom	2
Asymptotic Sig. (2-sided test)	.003

1. The test statistic is adjusted for ties.

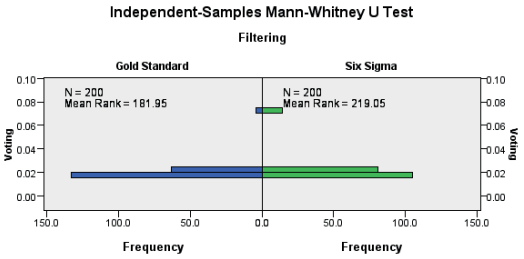
Figure 5.9: The Kruskal-Wallis ANOVA test shows a significant difference between the three voting data sets.

Finally, the performance of the voting approach is evaluated. Figure 5.9 presents the box plot evaluation of that aggregation mechanisms. According to the Kruskal-Wallis test, the data sets have a significant difference among each other. Since the visualization does not exactly show which of the sets outperforms the other, two pair-wise Mann-Whitney U test (figure 5.10 and 5.11) follows in order to reveal the best combination in terms of quality. The evaluation provides the insight that the Six Sigma and Plain data sets have no significant difference among each other but both of them are outperformed significantly by the Gold Standard filtering – once again.



Total N	400
Mann-Whitney U	19'183.500
Wilcoxon W	39'283.500
Test Statistic	19'183.500
Standard Error	1'156.143
Standardized Test Statistic	-.706
Asymptotic Sig. (2-sided test)	.480

Figure 5.10: The Mann-Whitney-U test shows that the voting aggregation mechanism applied on the Six Sigma and plain data set does not differ significantly.



Total N	400
Mann-Whitney U	16'290.000
Wilcoxon W	36'390.000
Test Statistic	16'290.000
Standard Error	1'156.143
Standardized Test Statistic	-3.209
Asymptotic Sig. (2-sided test)	.001

Figure 5.11: The Gold Standard filtering outperforms the voting application on the Six Sigma data set - the Mann-Whitney U test shows a significant difference.

5.1.2 Data Quality

This section shows the results of the several statistical evaluations for the *Data Quality* experiment. It shows the overall comparison of the *output agreement* aggregation method, as explained in section 3.2.3, as well as the *voting* aggregation (explained in the same section). But first, a line chart diagram (figure 5.12) shows the quality progress depending on the number of inputs. The chart visualizes that the *Six Sigma - output agreement* improves with increasing numbers of input, namely with 19% when comparing a single input with the maximum of 10 inputs. The Levenshtein distance has a value of .067 (*sd* = .004), whereas a value of 0 would indicate a perfect result. The *Six Sigma - voting* combination shows an improvement of 0.15% and a Levenshtein distance of .085 (*sd* = .006). The *Gold Standard - output agreement* and *Gold Standard - voting* approach deteriorated by 1.5% and 11%, respectively. At the basis of 10 inputs, they had a Levenshtein distance of .065 (*sd* = .0045) and .084 (*sd* = .0049). It must be noted that all these values are very close to zero and due to the overall quality of these methods and aggregation combinations, they provide very accurate results.

The Kruskal-Wallis test shows that the 6 different combinations differs significantly from each other, whereas the *output agreement* seems to outperform the *voting* approach as figure 5.13 presents. Since the box plot diagram in figure 5.14 does not show at a first glance which combination outperforms the others, three pair-wise Whitney-Mann U tests were also executed, see figure 5.15, 5.16 and 5.17. The Mann-Whitney U tests reveal that the *Gold Standard* filtering approach outper-

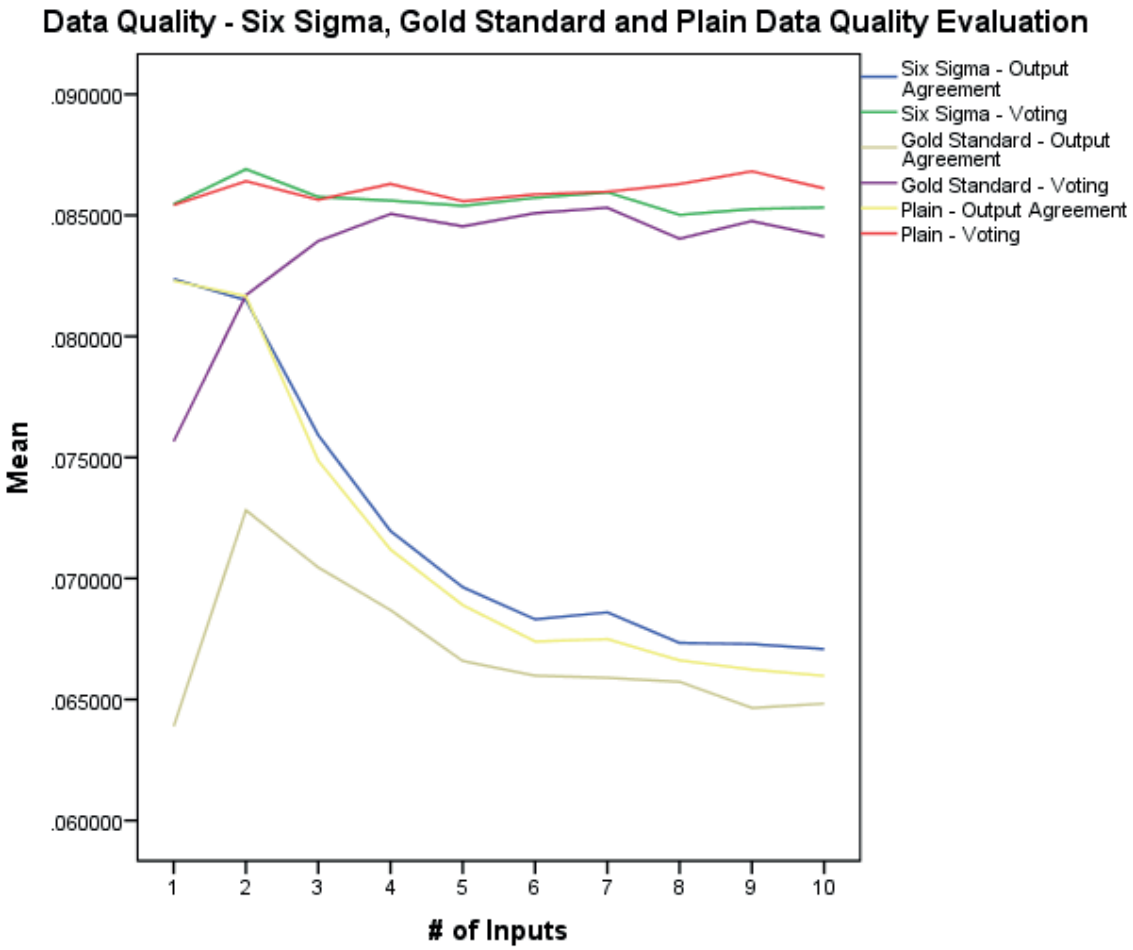
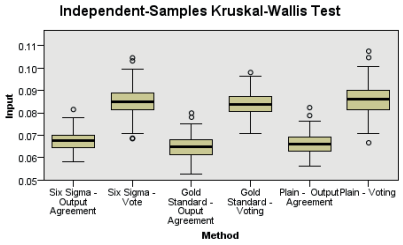


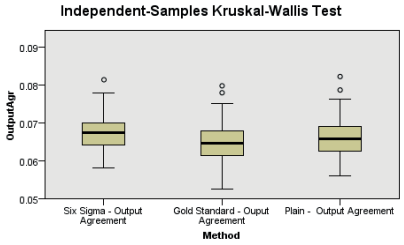
Figure 5.12: Data Quality - Changes in quality with increasing numbers of answers. Quality is calculated by Levenshtein distance (section 4.2.2), where a value of 0 is perfect and 1 is bad.



Total N	1'200
Test Statistic	890.167
Degrees of Freedom	5
Asymptotic Sig. (2-sided test)	.000

1. The test statistic is adjusted for ties.

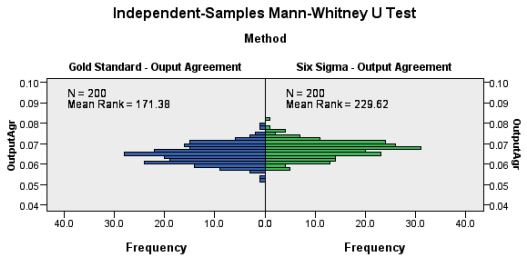
Figure 5.13: Statistical evaluation of all filtering-aggregation combinations by applying a Kruskal-Wallis ANOVA test to the Data Quality experiment. It shows that they differ from each other significantly with a p-value of .01.



Total N	600
Test Statistic	25.635
Degrees of Freedom	2
Asymptotic Sig. (2-sided test)	.000

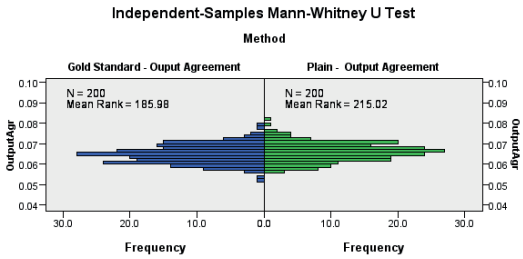
1. The test statistic is adjusted for ties.

Figure 5.14: Statistical evaluation of the output agreement aggregation method by applying a Kruskal-Wallis ANOVA test. It shows a significant difference between these 3 sets with a p-value of .01.



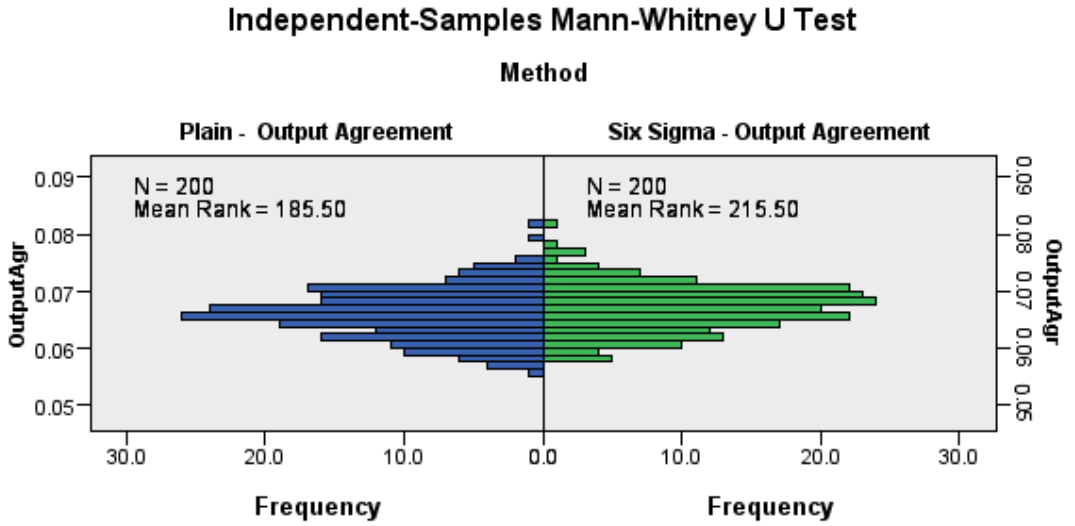
Total N	400
Mann-Whitney U	14'175.500
Wilcoxon W	34'275.500
Test Statistic	14'175.500
Standard Error	1'156.143
Standardized Test Statistic	-5.038
Asymptotic Sig. (2-sided test)	.000

Figure 5.15: Mann-Whitney-U test for Gold Standard and Six Sigma output agreement sets, Data Quality experiment.



Total N	400
Mann-Whitney U	22'904.000
Wilcoxon W	43'004.000
Test Statistic	22'904.000
Standard Error	1'156.143
Standardized Test Statistic	2.512
Asymptotic Sig. (2-sided test)	.012

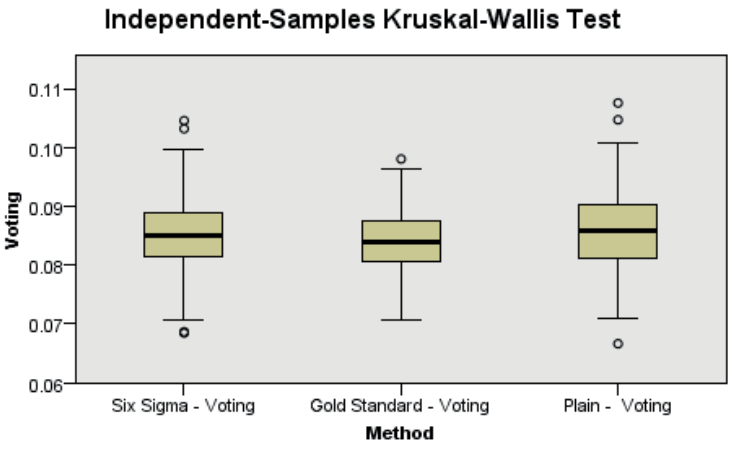
Figure 5.16: Mann-Whitney U test for Gold Standard and Plain output agreement sets, Data Quality experiment.



Total N	400
Mann-Whitney U	17'000.000
Wilcoxon W	37'100.000
Test Statistic	17'000.000
Standard Error	1'156.142
Standardized Test Statistic	-2.595
Asymptotic Sig. (2-sided test)	.009

Figure 5.17: Mann-Whitney U test for Plain and Six Sigma output agreement sets, Data Quality experiment.

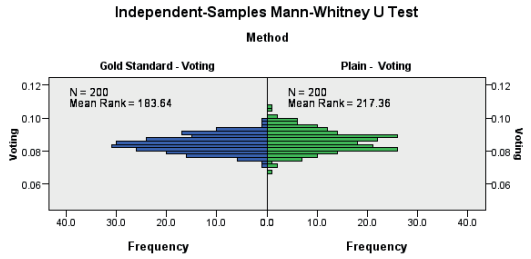
forms the other two data sets significantly. A surprising finding however is the fact that *Six Sigma* looses against the *Plain Data* set (figure 5.17) - which means that the filtering method filtered out results which were good but not within the range (see 3.1.1 for more details) whereas the plain data set considered all available answers.
The Kruskal-Wallis test does also not immediately show which filtering method in combination with the *voting* mechanism provides the best output but the evaluation shows that there is a significant difference (figure 5.18). Thus, two Mann-Whitney U tests were executed in order to analyze the sets pair-wise. Figure 5.19 and 5.20 present that the *Gold Standard* set outperforms the other two significantly in terms of quality.



Total N	600
Test Statistic	9.184
Degrees of Freedom	2
Asymptotic Sig. (2-sided test)	.010

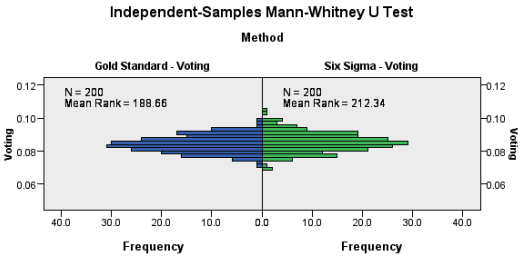
1. The test statistic is adjusted for ties.

Figure 5.18: Kruskal-Wallis test for voting mechanism, Data Quality experiment.



Total N	400
Mann-Whitney U	23'372.000
Wilcoxon W	43'472.000
Test Statistic	23'372.000
Standard Error	1'156.143
Standardized Test Statistic	2.917
Asymptotic Sig. (2-sided test)	.004

Figure 5.19: Mann-Whitney U test for Gold Standard and Plain voting sets, Data Quality experiment.



Total N	400
Mann-Whitney U	17'632.000
Wilcoxon W	37'732.000
Test Statistic	17'632.000
Standard Error	1'156.143
Standardized Test Statistic	-2.048
Asymptotic Sig. (2-sided test)	.041

Figure 5.20: Mann-Whitney U test for Gold Standard and Six Sigma voting sets, Data Quality experiment.

5.1.3 Rewriting

The third experiment was about rewriting phrases - the collected answers were filtered by the *Six Sigma* and *Gold Standard* mechanisms and then aggregated by an *output agreement* or a subsequent *voting* mechanism. The following evaluation results will highlight which approach outperformed the others and show how the quality changes with increasing numbers of inputs. It is important to note that the applied evaluation metric *Meteor* (explained in 4.2.3) considers a value of 1 as perfect and values closer to 0 as worse, this is the opposite as seen in the other two sections where the evaluation metrics consider a value closer to 0 as more accurate. Since only one reference sentence per task was collected in the Rewriting experiment, whereas the evaluation metric *Meteor* could handle multiple references as perfect translations, the overall metric indicates low quality output. However, *Meteor* is rather strictly and a high metric value is not easy to achieve. Figure 5.21 present that only the *Six Sigma - output agreement* combination has improved with increasing numbers of input, namely by 1.45% and had a Meteor value of .3296 (*sd* = .0026). *Six Sigma - voting*, *Gold Standard - output agreement* and *Gold Standard - voting* changed to the worse by .25%, 5% and 4.2%. The calculated Meteor values were .2864 (*sd* = .0034), .34 (*sd* = .0087) and .304 (*sd* = .021).

Whether all possible combinations differs from each other significantly will be disclosed by the following tests, beginning with a global Kruskal-Wallis evaluation. It seems, according to figure 5.22 that the voting aggregation mechanisms is defeated by the output agreement approach. The

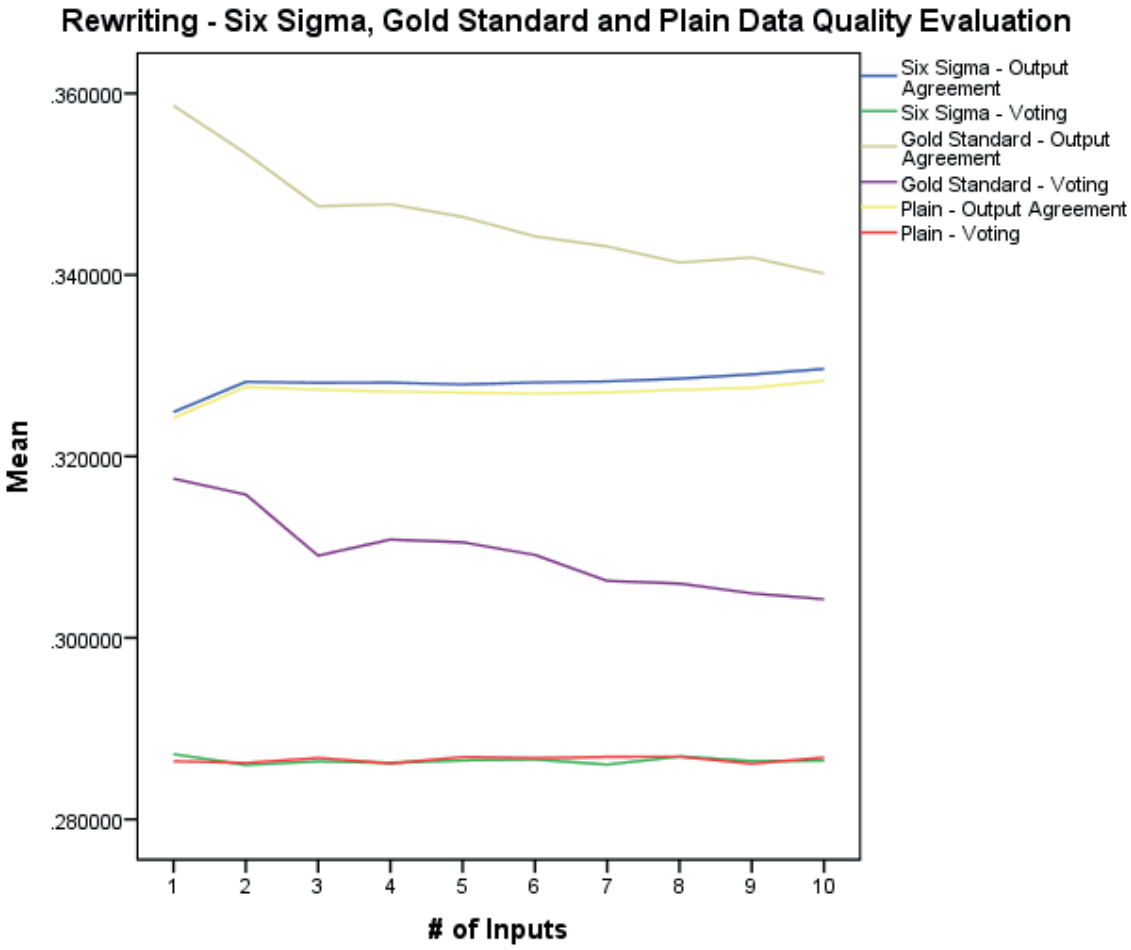
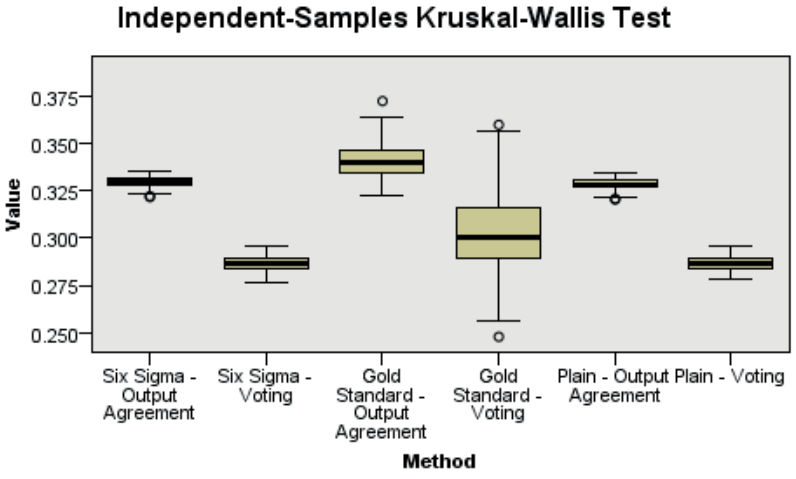


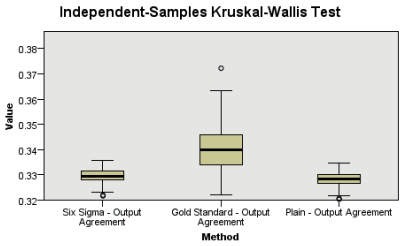
Figure 5.21: Rewriting - Changes in quality with increasing numbers of answers. Quality is calculated by Meteor, where a value of 1 is perfect and 0 is bad.



Total N	1'200
Test Statistic	898.888
Degrees of Freedom	5
Asymptotic Sig. (2-sided test)	.000

1. The test statistic is adjusted for ties.

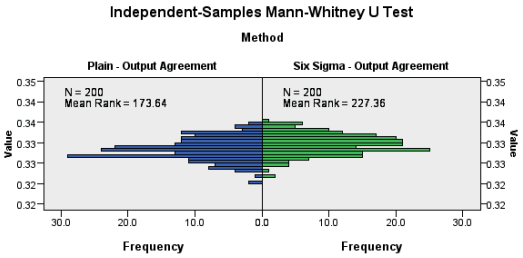
Figure 5.22: Overview of both combination possibilities for the Rewriting experiment with significant difference between the different sets.



Total N	600
Test Statistic	264.633
Degrees of Freedom	2
Asymptotic Sig. (2-sided test)	.000

1. The test statistic is adjusted for ties.

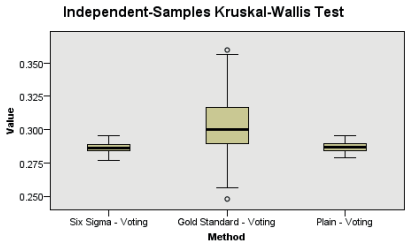
Figure 5.23: Kruskal-Wallis test for the output agreement data sets within the Rewriting task.



Total N	400
Mann-Whitney U	14628.000
Wilcoxon W	34728.000
Test Statistic	14628.000
Standard Error	1156.142
Standardized Test Statistic	-4.646
Asymptotic Sig. (2-sided test)	.000

Figure 5.24: Rewriting task with output agreement - Mann-Whitney U test for Plain and Six Sigma.

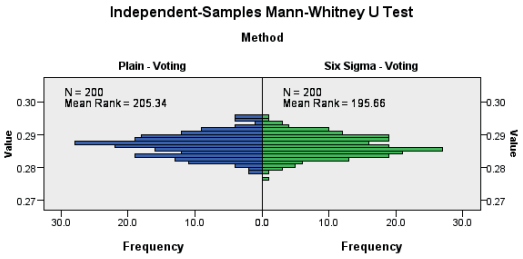
in-depth Kruskal-Wallis test in figure 5.23 shows whether among the output agreement data sets also some significance difference can be found. According to that figure the *Gold Standard - output agreement* is significant better than the *Six Sigma* approach as well as the base line with the plain data set. However, at least the *Six Sigma* approach outperforms the plain data set significantly, as evaluated by a Whitney-Mann U test in figure 5.24. The same analysis was conducted for the voting aggregation mechanisms whereas the box plot output of the Kruskal-Wallis test in figure 5.25 shows that the *Gold Standard* filtering in this scenario as well outperformed the other two significantly. A pair-wise Mann-Whitney U test (figure 5.26) reveals that the *Six Sigma* data set does not significantly differs from the base line.



Total N	600
Test Statistic	142.380
Degrees of Freedom	2
Asymptotic Sig. (2-sided test)	.000

1. The test statistic is adjusted for ties.

Figure 5.25: Kruskal-Wallis test for the voting data sets within the Rewriting task.



Total N	400
Mann-Whitney U	20'967.000
Wilcoxon W	41'067.000
Test Statistic	20'967.000
Standard Error	1'156.143
Standardized Test Statistic	.836
Asymptotic Sig. (2-sided test)	.403

Figure 5.26: Rewriting task with voting - Mann-Whitney U test for Plain and Six Sigma.

5.2 Time and Costs

Besides of the performance in terms of quality, the spent amount of time and money are other important characteristics in order to formulate a complete statement about the overall performance. This section will compare whether the *Gold Standard* assurance mechanisms consumes significantly more time and money than the *Six Sigma* approach. This would make sense since the Gold Standard setup – as recalled from section 3.1.2 – requires additional control questions in each HIT and therefore it is necessary to publish more assignments in order to collect the same amount of answers as with the Six Sigma filtering method.

As table 5.1 summarizes, it took, for example, 160 seconds on average to collect 10 estimations for a displayed pie chart diagram. In order to collect 10 business domain names, it took 10 different workers almost 10 minutes or 598 seconds.

This values are true for the Six Sigma approach but when applying the Gold Standard technique, the costs in terms of time are significant higher. The process of collecting 10 pie chart size values took 289 seconds on average because 1 out of 3 tasks was a gold question. Collecting business information with the same approach was also significant slower. On average, a requester had to wait 1212 seconds until 10 business URLs were found for a given business address.

	$PieChartEst.$		$DataQuality$		$Rewriting$	
	SS	GS	SS	GS	SS	GS
$mean$	160	289	598	1212	2585	5157
max	172	339	642	1316	2723	9135
min	144	246	515	1059	2459	2618
sd	5.6	20.02	20.58	57.89	45.99	1051
$significant(t)$	yes		yes		yes	
$costs(\$)$	.183	.275	.22	.37	.55	1.1

Table 5.1: Overview of spent time in seconds per 10 collected answer sets for a HIT of each of the three conducted experiments. *SS* stands for *Six Sigma* and *GS* for *Gold Standard*. In each case, the Six Sigma method is significant faster and the spent amount of money is as well lower compared to the Gold Standard assurance mechanism.

6

Discussion

The evaluation presented in chapter 5 reveals some interesting insights about the quality assurance of both filtering methods Six Sigma and Gold Standard. A practical overview of the evaluation results is shown in table 6.1. The table presents the results of the statistical comparison in measures of quality, time and costs. In the case of the quality performance, the results are split into sub categories in order to match the aggregation mechanism combinations.

According to the quality evaluation performed in chapter 5, the Gold Standard approach outperformed the Six Sigma method as well as the base line in every case with one exception. In terms of quality, the Gold Standard assurance mechanism seems to be the best choice. However, it is surprising to see that in one specific case, namely when applying the median aggregation method – a classic statistical method – to the plain data, both filtering methods are outperformed significantly. This leads to the statement that when the underlying task is about numbers – in the case of this thesis it is about estimating sizes – a simple median aggregation applied to the raw data would provide the best available output.

Developing and using a Gold Standard method is nonetheless a recommendable way of controlling the quality if quality assurance has a high priority, because the evaluation shows that the Gold Standard approach provides significantly more accurate output in all other cases without any additional exception. When analyzing the quality assurance performance of the Six Sigma approach, one comes to the understanding that there are situations were it cannot outperform the plain data significantly - this is the case in the voting scenarios for the Pie Chart and Rewriting tasks.

However, if the focus lies on performance measures such as time and costs, the drawback of the Gold Standard method is revealed. In every single scenario, the Six Sigma and plain data sets provided output which was collected significant faster and cheaper than with the implementation of the Gold Standard quality assurance mechanism.

Measures	Task		Six Sigma	Gold Standard	Plain Data	p-value
Quality	Pie Chart	Aggr.				
		Average	-	+	-	.01
		Median	=	=	+	.05
		Mode	-	+	-	.01
		Voting	=	+	=	.01
	Data Quality	Output Agr.	-	+	-	.05
		Voting	-	+	-	.05
	Rewriting	Output Agr.	-	+	-	.01
		Voting	=	+	=	.01
Time	Pie Chart		+	-	+	.05
	Data Quality		+	-	+	.05
	Rewriting		+	-	+	.05
Costs	Pie Chart		+	-	+	.05
	Data Quality		+	-	+	.05
	Rewriting		+	-	+	.05

Table 6.1: Summary of all performance results for each conducted experiment. + indicates a significant better version, - a significant worse version and = means that no significant difference was found between these two methods.

6.1 Limitations and future work

The evaluation and statistical tests of the experiment and aggregation combinations were conducted on the maximum level of available output – this was in all cases the amount of 10 answers per task – collected for each task.
Thus, it would be interesting to see whether it makes any difference how many answers are taken into consideration. Maybe it could be sufficient to collect only 5 or 6 answers in some scenarios in order to get a significant better output than compared to other filtering and aggregation methods. In addition to this, future work may have to consider other error levels for accepting Gold Standard answers. This thesis, as described in 3.1.2, set the error level at .2 for the Pie Chart Estimation task as well as for the Data Quality task and at .3 for the Rewriting task. One could investigate whether it does influence the performance in terms of time and costs significantly when the error

level is very strictly.
All experiments were also designed in such a way that not every type of answer was accepted. For example, when the task was asking for a number then it was not possible to type in an answer which contained a character or a word. This settings however cost time whereas it is not clear whether maybe it would have been possible to provide simple standard answer fields without loosing any quality.
Another limitation occurs in the context of the evaluation of the rewriting experiment. The evaluation method *Meteor* uses reference sentences in order to calculate the quality of a translation – this work only provided one reference per assignment whereas *Meteor* would be capable of considering multiple reference sentences. This would lead to a more robust and probably higher evaluation rate. An evaluation in the same context may use more than one reference in order to maximize this evaluation method. Six Sigma and Gold Standard are of course not the only available quality assurance mechanisms – a future work could, therefore, evaluate a wider range of such techniques in order to cover the current state of the research.

7

Conclusions

Computers and the Internet changed our life significant. Beside of many other things, they enabled solutions and platforms in order to use the cognitive power of thousands of human beings. Crowdsourcing platforms like Amazon's Mechanical Turk helps to solve problems and tasks which are still very challenging for computers. A small incentive – in the context of crowdsourcing markets it is mostly a monetary incentive – is enough to motivate hundreds or even thousands of workers to categorizing photos, translating phrases, collecting data or estimating needed numbers.

However, monetary incentives attract also individuals who try to maximize their income with minimal effort – which means, they try to cheat by providing random answers or using machine tools in order to earn as much as possible. Furthermore, there are individuals who may not understood the task correctly and provide bad answers without any bad intention.

Since these workers are distributed all over the world, a classic hiring process is not possible and would cost too much time and money. Thus, it is necessary to apply other techniques in order to detect workers who provide bad quality.

Researchers explore this field for some time now and provided already interesting quality control techniques. However, the evaluation of these techniques focus in most cases only on the quality aspect – whereas it would be interesting to analyze in addition to this other economic factors such as time and costs as well. This would lead to a practical guideline for choosing the right quality assurance mechanism depending on the priorities and the task.

This thesis analyzed and compared for two quality assurance methods – the Six Sigma filtering as well as the Gold Standard – their performance in terms of quality, time and costs by conducting three experiments.
The evaluation of the collected data sets provided the insights in order to answers the questions whether *(1)* the Six Sigma filtering method outperforms the Gold Standard method in terms of quality, time and costs, whether *(2)* both filtering methods provide always better quality compared against a base line and whether *(3)* the quality assurance mechanisms always provide robust output, regardless of the underlying task.

The evaluation revealed that the Six Sigma method does not outperform the Gold Standard mechanism in terms of quality but in terms of time and costs. The results of the experiments showed that a filtering method, particularly the Gold Standard method, provides significant better output than unfiltered, raw data sets – with the exception of one scenario where a median aggregation method in combination with the plain data outperforms both filtering methods. In addition to this, it is also important to note that the Six Sigma approach was not able to outperform the base line in every case. Since the quality assurance mechanisms were applied in three different use cases, it is also possible to say that the underlying task does not influence the performance. The Gold Standard technique was found to be particular robust.

List of Figures

List of Tables

List of Listings

Bibliography

[Bernstein et al., 2012] Bernstein, A., Klein, M., and Malone, T. W. (2012). Programming the global brain. *Commun. ACM*, 55(5):41–43.

[Bernstein et al., 2010] Bernstein, M. S., Little, G., Miller, R. C., Hartmann, B., Ackerman, M. S., Karger, D. R., Crowell, D., and Panovich, K. (2010). Soylent: a word processor with a crowd inside. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pages 313–322, New York, NY, USA. ACM.

[Brabham, 2008] Brabham, D. C. (2008). Crowdsourcing as a model for problem solving: An introduction and cases. *Convergence*, 14(1):75.

[Craig Eric Schneier et al., 1995] Craig Eric Schneier, P., Shaw, D., Beatty, R., and Baird, L. (1995). *The Performance Measurement, Management, and Appraisal Sourcebook*. Human Resource Development Press.

[Denkowski and Lavie, 2011] Denkowski, M. and Lavie, A. (2011). The meteor automatic mt evaluation metric,. `http://www.cs.cmu.edu/~alavie/METEOR/`.

[Doan et al., 2011] Doan, A., Ramakrishnan, R., and Halevy, A. Y. (2011). Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96.

[Galton, 1907] Galton, F. (1907). The Ballot-Box. *Nature*, 75(1952):509–510.

[Howe, 2006a] Howe, J. (2006a). Crowdsourcing: A definition.

[Howe, 2006b] Howe, J. (2006b). Wired 14.06: The Rise of Crowdsourcing.

[Ipeirotis and Horton, 2011] Ipeirotis, P. G. and Horton, J. J. (2011). The need for standardization in crowdsourcing. CHI 2011.

[Ipeirotis et al., 2010] Ipeirotis, P. G., Provost, F., and Wang, J. (2010). Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 64–67, New York, NY, USA. ACM.

[Kern et al., 2010a] Kern, R., Thies, H., Bauer, C., and Satzger, G. (2010a). Quality assurance for human-based electronic services: A decision matrix for choosing the right approach. In *ICWE Workshops*, pages 421–424.

[Kern et al., 2010b] Kern, R., Thies, H., and Satzger, G. (2010b). Statistical quality control for human-based electronic services. In *ICSOC*, pages 243–257.

[Lavie and Agarwal, 2007] Lavie, A. and Agarwal, A. (2007). Meteor: an automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07, pages 228–231, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Law and von Ahn, 2011] Law, E. and von Ahn, L. (2011). *Human Computation (Synthesis Lectures on Artificial Intelligence and Machine Learning)*. Morgan and Claypool Publishers, 1 edition.

[Levenshtein, 1966] Levenshtein, V. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8):707–710. Original in *Doklady Akademii Nauk SSSR* 163(4): 845–848 (1965).

[Malone et al., 2010] Malone, T., Laubacher, R., and Dellarocas, C. (2010). The collective intelligence genome. *Engineering Management Review, IEEE*, 38(3):38 –52.

[Minder and Bernstein, 2011] Minder, P. and Bernstein, A. (2011). Crowdlang - first steps towards programmable human computers for general computation. In *Proceedings of the 3rd Human Computation Workshop*, AAAI Workshops, pages 103–108. AAAI Press.

[Minder and Bernstein, 2012] Minder, P. and Bernstein, A. (2012). How to translate a book within an hour - towards general purpose programmable human computers with crowdlang. In *Web Science 2012*, New York, NY, USA.

[Nohel et al., 2012] Nohel, J., Sattinger, D., and Rota, G.-C., editors (2012). *Selected Papers of Norman Levinson: Volume 2 (Contemporary Mathematicians)*. Birkhaeuser, softcover reprint of the original 1st ed. 1998 edition.

[Paolacci et al., 2010] Paolacci, G., Chandler, J., and Ipeirotis, P. G. (2010). Running experiments on amazon mechanical turk. *Judgment and Decision Making*, 5(5):411–419.

[Quinn and Bederson, 2011] Quinn, A. J. and Bederson, B. B. (2011). Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1403–1412, New York, NY, USA. ACM.

[Rojas, 1983] Rojas, R. (1983). Die Rechenmaschine von Konrad Zuse - Sechzig Jahre Computergeschichte. *Spektrum der Wissenschaft*.

[Sun and Dance, 2012] Sun, Y.-A. and Dance, C. R. (2012). When majority voting fails: Comparing quality assurance methods for noisy human computation environment. *CoRR*, abs/1204.3516.