Anomaly Detection of Wikipedia Traffic Data

AIM-3 Project Report

Felix Neutatz
TU Berlin
neutatz@campus.tu-berlin.de

Stephan Alaniz Kupsch
TU Berlin
alanizkupsch@campus.tu-berlin.de

ABSTRACT

In this paper we analyze Wikipedia traffic data from 1 January 2014 until 31 July 2014 for detecting anomalies. Two different models are presented and evaluated. One incorporates two-sided moving averages and the other uses multiple linear regression. We detect an anomaly if the raw data differs from our models by a certain threshold. As threshold we use a method based on the three sigma rule. Manually evaluated Wikipedia traffic data shows that the regression model outperforms the moving average model in anomaly classification accuracy. It is possible to relate real world events to Wikipedia traffic anomalies. We also discover strange behavior/anomalies in the data that cannot be easily traced back to its origins. Our approach can be easily extended to a streaming use case.

1. INTRODUCTION

Anomaly detection is the "Sherlock Holmes" of machine learning. It is spotting the outliers - out of common events or patterns, finding abnormal trends and detecting fraud. There is a wide range of use cases for anomaly detection. It can be applied in financial applications like monitoring the stock market [10], in geology, recognizing earth quakes [8] and last but not least for marketing i.e. alerting the company when customers start to complain in social networks [11].

With the internet of things and the era of Big Data, there is an increasing interest in applying anomaly detection continuously on very large volumes of data in real time and automatically. But in the end it is the fascination to uncover the anomalies and solve the puzzle, which makes anomaly detection a hot topic in computer science.

This project was elaborated in the course "Advanced Information Management III Scalable Data Analysis and Data Mining" of Technische Universität Berlin in a timespan of 6 weeks. We analyze a time series of Wikipedia traffic data in order to find outliers. Ideally, these anomalies can be mapped to real world events. Our goal is to develop techniques and scalable models which can reliably find unusual behavior in the data. In the course of the project we first develop a simple algorithm based on moving averages and afterwards we go one step further by applying a more complex multiple linear regression model[1].

The rest of the paper is organized as follows: Section 2 defines the problem statement of this project. Section 3 presents the data and the preprocessing steps. Section 4 explains the environment and systems of our analysis. Section 5 describes the algorithms and models we used for anomaly detection. Section 6 shows our experiments and discusses our results on the Wikipedia traffic data. We conclude our work with respect to further ideas and applications to which this project can be extended in Section 7.

2. PROBLEM STATEMENT

The goal of the project is to find anomalies and interesting structure in the raw Wikipedia traffic data. Given a subset of the data, e.g. a single Wikipedia page, we analyze the time series and try to find relations to real world events and cultural movements. We validate our results manually for a few Wikipedia pages. We discuss these findings and conclude consequences to improve our approach gradually. Finally, we target to develop an unsupervised learning approach that can detect dates or time periods that strongly differ from overall regression trends of Wikipedia traffic. Since we want this approach to work in a Big Data setting, we use the distributed system Apache Flink for processing the data. This project focuses on a batch processing approach. This means that we start from the premise that all data is collected before conducting the data analysis.

3. DATA & PREPROCESSING

This section describes the structure of the Wikipedia traffic data and the steps to choose a reasonable subset of the data. This helps us to prototype rapidly and validate our results in time.

3.1 Data

The data used for this project consists of hourly page traffic logs from Wikipedia [7]. For every hour of traffic statistics, there is one file. Each line in the data represents a Wikipedia page which is identified by the language code and its title. This is followed by the total count of visits

[1]Github repository of this project's implementation: `https://github.com/FelixNeutatz/wikiTrends`

Table 1: Example lines from the data of one hour

Lang.	Page	Visits	Traffic (Byte)
de	Angela_Merkel	50	4350435
...	...	...	...
en	Barack_Obama	480	147172483
en	Barack_Obama,_Sr.	31	1061450
...	...	...	...
en	Facebook	1373	220402148

to this page during this hour and the traffic generated by the page in bytes. Table 1 shows an example of this data format.

Collecting these statistics was initially started by Domas Mituzas in 2007. The data records visits (not unique visits) of every single Wikipedia page up to date. This project analyzes Wikipedia traffic statistics from 1 January 2014 until 31 July 2014, a total of 7 months. For this data set, the size of hourly traffic statistics ranges on average between 80 MB and 110 MB depending on the time of the day. This sums up to over 2 GB per day, almost 70 GB per month and 475 GB for the full 7 months of data. Unfortunately, the data lacks records of 5th to 6th January. This slightly influences the results of our anomaly detection experiments.

The data is highly compressed and the raw data is about 3.5 times bigger than the presented figures. The original scope of this project aimed to analyze at least a whole year of Wikipedia traffic data. Unfortunately, the computer cluster we used did not allow to make this analysis for more than 7 months of data. The cluster reached disk space limitations and suffered from slow download speed, so that downloading the 475 GB of data already took two and a half weeks of non-stop downloading.

3.2 Preprocessing

Preprocessing is an important step in analyzing the traffic data from Wikipedia since processing the data all at once leads to some problems. The sheer amount of uncompressed data is too big for some of the calculations. Even when analyzing the data on a distributed system like Apache Flink the processing of the data reaches hardware limitations (see Section 5.2.2 for details). Therefore, the data needs to be preprocessed, i.e. cleaned and reduced to a subset that is both processable and analyzable in a reasonable amount of time.

First of all, we filter the data to keep pages written in the English language (language code: "en") and written in the German language (language code: "de") since these pages interest us personally. It makes it easier for us to validate anomalies found in these two languages.

Furthermore, in order to choose a suitable subset of English and German Wikipedia data, we analyzed the distribution of average page visits per day. Figure 1 shows a histogram of average page visits per day, showing only average visits between 5 and 100. We emit pages outside this range for visualization purposes. The figure clearly shows the heavy tailed distribution of pages, where many pages exhibit a very small amount of visits whereas only few pages are actually often visited. Among the most visited pages are special Wikipedia related pages like the "Main Page" or the "Search". We now subset the data by picking only the top k pages for both English and German according to the average visits per day statistic. After experimenting on a small data set of the top 100 pages, we increase the data set size to top 10,000 German pages and top 50,000 English pages. This corresponds to approximately 0.15% of the total pages per language.

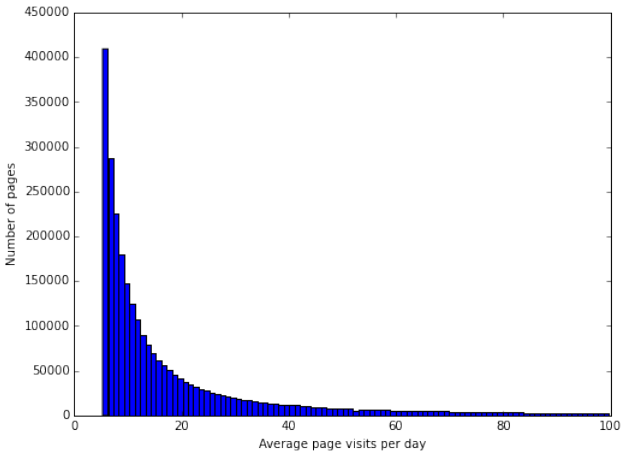

Figure 1: Histogram of average visits per day ("en" and "de" pages), showing only average visits between 5 and 100

In this project we are only interested in the number of visits. We do not discuss the number of transferred bytes per page. We make this choice because page visits better refer to interactions of humans whereas the transferred bytes depend on the page's content size.

4. TECHNOLOGY

4.1 Flink

In this project we use Apache Flink [5] as data processing system. Flink has a streaming runtime which leverages in-memory processing where possible e.g. pipelining. Moreover Flink provides native iterations which accelerate iterative processing for big data as we need it in the context of this project. Besides a very fast runtime it also enables the user to connect to file systems like HDFS, the local filesystem and many more. Another advantage of Flink is that it is really easy to use ;-). There is huge variety of operators like map, reduce, filter and join which make it more convenient to develop in the big data world. Being early adopters, we also use the newly released machine learning library to make this project a success. Also in consideration of continuing this project, Flink is a good choice. Since anomaly detection is a very good fit for stream processing, Flink enables a close to seamless integration from batch to streaming jobs. Moreover in contrast to Apache Spark, Flink provides real streaming and not discretized streaming[13]. This allows users more freedom in defining windows. More ideas on putting this project into the world of real time streaming will be further elaborated in the Section 7.

4.2 Cluster environment & Setup

We use a 10-node Ubuntu cluster. Each node has 48 IBM Power7 CPUs and 50GB of main memory. Moreover we apply a snapshot version of Flink 0.9 in order to be able to run it with IBM Java. We implemented a quick workaround

to make this happen. The bug has been reported and will be fixed in the upcoming Flink release. We run Flink with parallelism of 10.

5. ANOMALY DETECTION

We define anomalies in the Wikipedia traffic data as hours or days at which the traffic skyrockets against the usually expected average or trend of a page. That means a page that is on a constant trend of becoming more popular is not detected as an anomaly unless a particular moment stands out of the usual trend. The goal of this section is to describe our approach in finding these anomalies and present the models we use to automatically calculate outliers without prior knowledge. The anomalies are detected per page. This means that the presented models operate on one page at a time.

We refer to a trend when we observe an overall increase or decrease of the average visits of a page. Seasonality is defined as recurring patterns in the data in a fixed period of time. For instance, you can assume that German pages have more frequent page visits during the day as compared to night time. This is one example of a daily seasonality.

5.1 General Approach

The approach to find anomalies can be described as follows:

1. Fit a model to the data. This model describes the average/trend of the data at any point in time.
2. Subtract the model from the raw data in order to eliminate seasonality and trend. This leaves residuals to the model for each time stamp.
3. Apply a threshold to the residuals. An observation is classified as anomaly if its absolute residual is greater than the threshold.

This unsupervised procedure is done for every single page to detect anomalies from.

Our first approach is to take a moving average model. This is a rather simple model which captures a general trend of the data. However, the model is not sophisticated enough to capture seasonality. Therefore, we introduce a second approach which uses a multiple linear regression model. A more detailed description of these approaches can be found in Sections 5.2 and 5.3.

5.1.1 *Quantile threshold*

After calculating the residuals with one of the models, we need to decide on a threshold to classify anomalies. Ted Dunning et al [3] suggest to calculate a quantile of the residuals. The k% quantile defines the number x such that the probability, that a number from the same distribution is greater than x, is at most $100\% - k\%$. After calculating the quantile of the residuals, we filter any residual that is greater than the quantile. This means that the most extreme residuals remain. For example, taking the 99% quantile of the residuals returns a value which is greater than 99% of all residuals. The remaining residuals, i.e. 1% of all residuals, are identified as anomalies. It is suggested to take a quantile percentage which gives a number of anomalies equal to the maximum number of false positives one is willing to accept. This approach classifies anomalies even if there are no outliers at all in the data. Therefore, anomalies would have to be manually checked by a human in order to ensure high quality results.

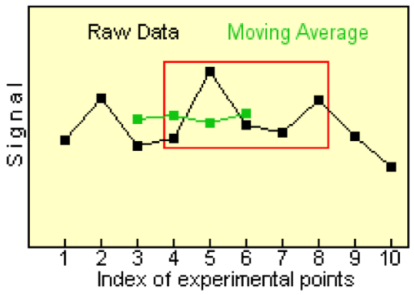

Figure 2: Moving Average with window size 5 (red) after calculating 4 smoothing averages (green) [4]

Ted Dunning et al [2] describe how to calculate quantiles in an online fashion. In the paper they introduce a distributed data structure called t-digest which helps to approximate the quantile using a minimal amount of memory. Therefore, this approach of determining the threshold is extremely well suited for a streaming use case.

Unfortunately t-digest classes are not yet implemented as *Serializables*. Therefore this currently works only locally. The project scope did not allow to implement this accordingly.

5.1.2 *Variance threshold*

We suggest a different way of choosing the threshold by using the variance in the residuals. The motivation from this approach originates from the so called "68-95-99.7 rule" (also known as "The three sigma rule" [9]) which describes the amount of data (in percentages) which is within 1-2-3 times of the standard deviation from the mean of normally distributed data. So when calculating the standard deviation of the residuals - assuming a normally distributed noise - we detect an anomaly if a residual is greater than 3 times the standard deviation. Statistically, this approach classifies 0.3% of the residuals as false positives and finds any outlier.

Equation 1 describes the process of detecting anomalies based on the "68-95-99.7 rule". x_t is the data point at time t, r_t is the residual of x_t and σ_t is the standard deviation of r_t. If the standard deviation is not calculated locally at time t, σ_t equals to the global standard deviation of r.

$$anom(x_t) = \begin{cases} 1 & \text{if } r_t \geq 3 * \sigma_t \\ 0 & \text{else} \end{cases} \quad (1)$$

The variance threshold is used in this project for all of the presented models.

5.2 Moving Average Model

The moving average model can be used to smooth raw time series data to a non-parametric function of averages [12]. Different variations of this model are often used in financial time series analysis among other applications.

5.2.1 *Algorithm*

The moving average algorithm calculates the average value of multiple subsequent subsets of the data which are captured in a window. Equation 2 shows the calculations for each window.

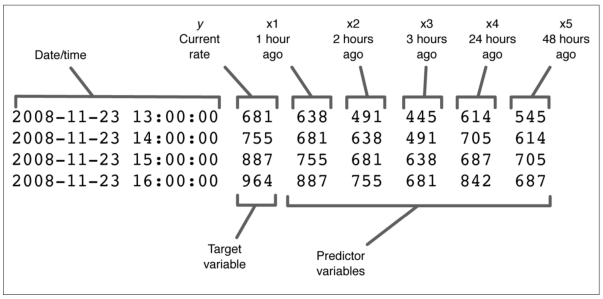

Figure 3: Training data format for regression [3]

$$MA_t = \frac{x_{1+t*s} + x_{2+t*s} + \ldots + x_{w+t*s}}{w} = \frac{1}{w}\sum_{i=1}^{w} x_{i+t*s} \quad (2)$$

The algorithm starts with the first w data points. w is hereby denoted as the window size. This average now serves as the function value of the moving average model at time t_0. This time stamp is usually defined to be either the median time stamp or the last time stamp of the data in the window. In literature [6] this is known as the two-sided moving average and the one-sided moving average respectively. The window of the first w data points is now moved forward by step size s, i.e. the first s data points of the window are removed and the next s data points after the current window are added. The same procedure of calculating the average is repeated for the new window. The window is moved forward until all the data in the time series is processed. Finally, when all function values of the moving average model have been calculated, the points are interpolated to get a function of moving averages.

A common practice is to use a slice size of one so that a new window is created by tossing one data point out and inserting the next data point into the window. This gives the smoothest moving average function one can achieve for a data set. It might still be useful to define a bigger slice size, for instance to improve performance when a little less detail in the moving average functions still gives reasonable results.

Figure 2 shows a small example of how the moving average works. The black points represent the raw data of a time series. A window (red rectangle) is drawn around the points which are used to calculate the current window average. The moving average of 3 previous windows as well as the moving average of the current window are shown in green. This version corresponds to the two-sided moving average where the calculated average is placed in the middle of the window as in Equation 2.

We hereby propose to calculate the variance alongside the moving average calculations for efficiency. Instead of calculating the variance over all residuals, the windows of the moving average allow to calculate the variance of these subsets simultaneously. Hence, we get both a moving average as well as a moving variance in this model. The moving variance is nice because it adjusts better to recent changes in the variance. For example, when there is suddenly a lot of fluctuation in the data, the algorithms will not classify every single peak of fluctuation as anomaly. It rather detects a high variance at local points in time and adjusts the anomaly threshold.

Table 2: Grid search for step size on main page traffic of Wikipedia

step size	sum of squared errors
5.0	4.43E34
3.0	3.38E23
1.0	157.68
0.5	191.52
0.3	209.97
0.1	234.40
0.05	322.84
0.03	568.58
0.01	1446.01

5.2.2 Flink Implementation

The Flink implementation of the moving average algorithm works just as described above on a distributed parallel system. There are, however, some considerations in how to fully exploit the parallelization capabilities of a distributed system such as Apache Flink. In general, since the algorithm is executed on every single Wikipedia page in our data set, the calculations of the moving average algorithm can be parallelized on the data level. Thus, one can think of the algorithm to work encapsulated on each Wikipedia page, which can be easily done simultaneously on a cluster with Flink.

We take a look at two different implementations of the moving average algorithm in Flink. The straight forward implementation of the algorithm reads the whole data set and iterates over the data set where in each iteration the moving average of one window is calculated. Even though, parallel systems usually do not excel at iterative algorithms, Flink has an efficient implementation which enables pipelining of iterations yielding better performance. The so called delta iteration in Flink allows the user to make changes to the data set that is processed (the window in each iteration) and to update a result set (the moving averages of each iteration). During the calculation of the moving average, the moving variance is calculated as well. The final result set of the iterations include all moving average and variance values calculated. Then, they are joined with the original data to calculate the residuals and finally the anomalies.

A different implementation involves data preprocessing by creating every single window's subset first and then executing the algorithm on all of the windows at once in parallel. This algorithm benefits not only in having data parallelism per page, but also per window. Thus, there is no need to have iterations. The drawback of this solution is that the data size is multiplied by the factor of how many windows a single data item is allocated to. For example, when the window size spans 7 days and for each window 1 day is replaced, the data size multiplies by 7 during execution. This may cause problems depending on the available memory. If this approach actually leads to swapping of data from memory to disk, it significantly slows down the program.

Our experiments show that using the second model leads to problems. We observe errors that nodes run out of memory and an error of "too many open files". We suspect that

Flink swaps a lot of the data to disk generating many temporary files which eventually lead to the problem that too many files are being opened simultaneously. We could not fully trace back this problem, though. Due to this problem we stick to the iterative implementation throughout this project.

5.3 Multiple Linear Regression Model

5.3.1 Algorithm

The second approach which we present in this paper uses multiple linear regression to approximate what is "normal" in the traffic of a page. The trend in addition to seasonality of a page is defined as "normal" in this case. In order to find a good model which shows what the norm in the traffic is, Ted Dunning et al [3] showed a practical approach how to do so. The idea is to train a model which is capable of reconstructing the data by training on the same data. For example that means predicting the traffic at hour t given that you know how many people visited the page one hour ago and two hours ago. In this way we can calculate the coefficients accordingly and find the best model which excels in this prediction in general.

In this project we use 5 features as described in figure 3: One hour ago, 2 hours ago, 3 hours ago, 24 hours ago and 48 hours ago. We use exactly theses features because they have already been shown to work in [3]. But feature engineering can definitely improve performance. This will be future work. The downside of this feature set is that we have to collect data for the first two days before being able to predict values and apply anomaly detection.

In this project we use the whole data of each page to train the model. This is ok, since we work on a batch application. But in the streaming use case you have to be smarter to find a representative data set to train on. Another approach would be to use online learning to adept to new trends. But this is again out of the scope of this paper and is not discussed here.

To train a model we use linear regression. In the general case the model of linear regression is described in the following way:

$$y = a^T x \tag{3}$$

In our case linear regression solves the following problem:

$$\begin{aligned} y_t = a_0 + a_1 * x_{t-1h} + a_2 * x_{t-2h} + a_3 * x_{t-3h} \\ + a_4 * x_{t-24h} + a_5 * x_{t-48h} \end{aligned} \tag{4}$$

such that the sum of squared residuals is minimized:

$$\arg\min_a \sum (y - a^T * x)^2 \tag{5}$$

Gradient descent is used to solve this optimization problem.

Once we have calculated the model (6 coefficients), we predict each data point according to the model. Now we have the original number of visits and the predicted number of visits (the norm) for every data point. As explained in chapter 5.1, we can calculate the residuals and apply either a quantile or a variance threshold.

We apply this approach per page. One could also think about training a model based on an additional feature which

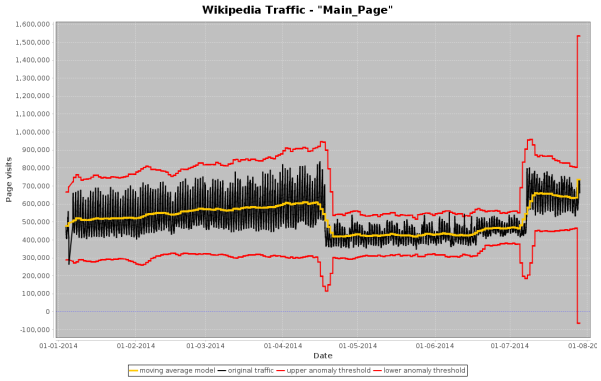

Figure 4: Anomaly detection of the main page of Wikipedia by Moving Average

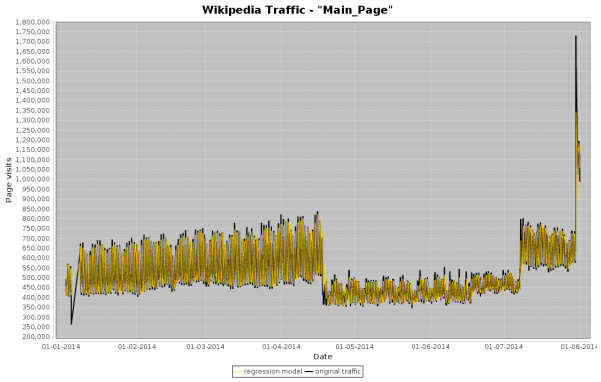

Figure 5: Regression model for main page of Wikipedia

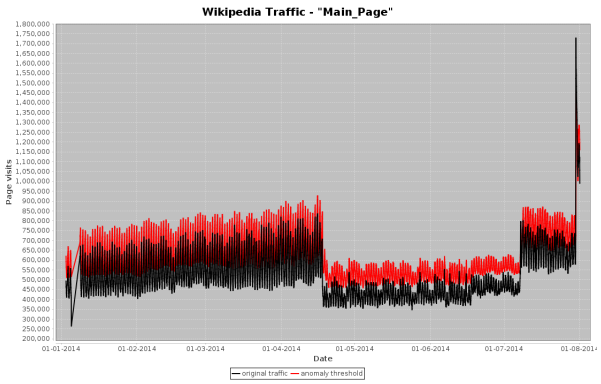

Figure 6: Anomaly detection of the main page of Wikipedia by Regression

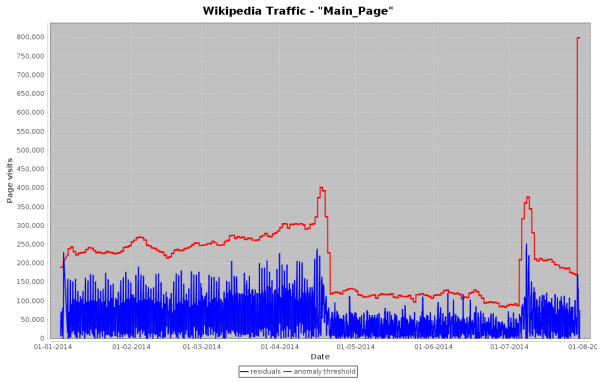

Figure 7: Anomaly detection of the main page of Wikipedia by Moving Average on the residuals

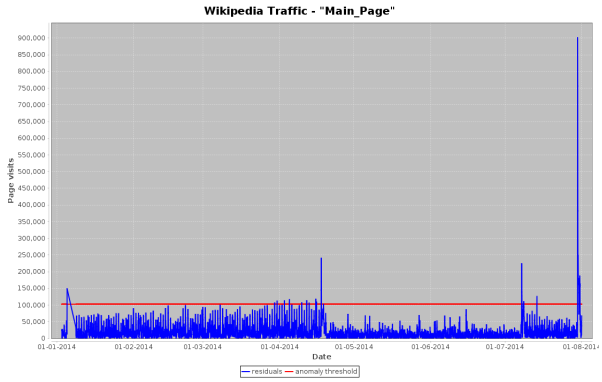

Figure 8: Anomaly detection of the main page of Wikipedia by Regression on the residuals

could be an average over all the data of all pages. Another idea would be to train a model per language. This could improve the fitting to regional seasonality.

5.3.2 *Flink Implementation*

The first step is to generate the feature representation described in 3. First, we generate an ID for every hour present in the data set. Then we apply basically 5 self joins. Only the hour ID is mapped corresponding to its feature, i.e. $x_t -> x_{t+1}$ for the first feature.

To prevent any numerical problems, we normalize the data set. Moreover we add a column of ones as bias to the data. Then we apply linear regression on the data set.

For the multiple linear regression we use the class *MultipleLinearRegression* from Flink ML. *MultipleLinearRegression* uses batch gradient descent to optimize Equation 5.

But there is also a problem with the implementation of *MultipleLinearRegression*. As input we get a *DataSet* of type (PageName,Data). So what we would like to do is group by the page name and apply the regression on this grouped data set. Unfortunately this is not possible at the moment, since *MultipleLinearRegression* does not support grouped data sets as input. Therefore we have to iterate over the pages. Since the gradient descent implemented in *MultipleLinearRegression* uses a native Flink iteration, we cannot use a Flink iteration to loop over the pages because nested iterations are not supported by Flink. This leads to the only viable solution that we unroll the loop by using a simple for loop like discussed in the Flink development mailing list[2].

Moreover using *MultipleLinearRegression* is not trivial because you have to tune the step size in order to achieve convergence using gradient descent. We use grid search for the right step size and take the model with the lowest squared residual sum (as shown in Section 6.1.2).

Another missing functionality of *MultipleLinearRegression* emerges when it comes to prediction. In the beginning, the data format looks like this: (hourID, LabeledVector(y,x)). But the training function *fit* needs the input in the form of LabeledVector(y,x) without the hourID. This means we loose the binding to the actual time of the data point and this is needed to calculate the residuals. So what we do is, we apply *predict* as you would do it usually, but then we join the original data set (hourID, LabeledVector(y,x)) with the predicted values to assign each data point its time again. This is suboptimal because this is really an unnecessary join. But at the moment there is no other way to do this[3].

6. EXPERIMENTS & RESULTS

We use the data described in Section 3 and both models from Section 5 to detect anomalies. In order to validate the results, we use manual human validation by both looking at the graphs and researching for real events or news behind detected anomalies. Before going into detail about some of our findings, we want to show the basic functionality of the two models with an example. For that first example we show the whole scope of plots including the raw data as well as the plots for each of the models. Later results will only show a selected subset of the plots due to space limitations. We include the whole set of plots including the raw data plots of all examples in the appendix.

6.1 Experimental Procedure

We apply our Flink implementation of the two models on our preprocessed data sets. First, we run our analysis on a smaller subset of the data containing the top 100 most visited pages in German and English language. This analysis focuses on finding the correct parameters for our models and trying out how the algorithms work in general. After that, the bigger data set containing roughly 0.15% of all German and English pages is used to run our analysis on and find anomalies.

Validation of the detected anomalies is done manually on a smaller subset of the data since the defined problem is an unsupervised learning problem, where we don't have data about what should have been classified as an anomaly. However, anomalies can easily revealed by a human simply by looking at the time series graph of the raw Wikipedia traffic data. Some examples of this validation step are described in the following sections.

6.1.1 *Determining Window & Slice Size for Moving Average*

[2] `http://apache-flink-mailing-list-archive.1008284.n3.nabble.com/Building-several-models-in-parallel-td6905.html`

[3] `https://www.mail-archive.com/dev@flink.apache.org/msg02931.html`

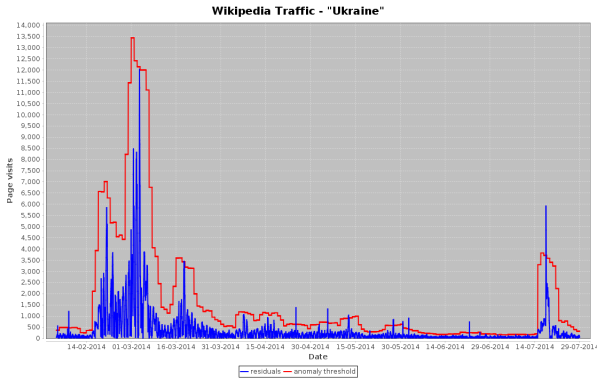

Figure 9: Residuals of the moving average model of the page "Ukraine"

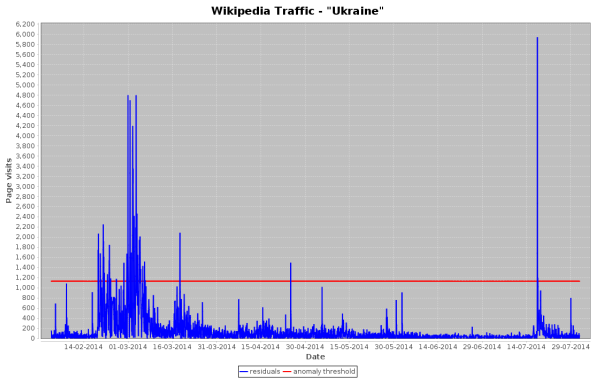

Figure 10: Residuals of the regression model of the page "Ukraine"

For the moving average model, we tested both the impact of the window size and the slice size. It turns out that the window size has a much higher impact on the results than the slice size. We tried out window sizes between 1 and 30 days and observed that small window sizes tend to overfit the data and therefore do not detect many of the anomalies. A high window size underfits the data and often classifies more anomalies than wanted since the model does not fit new trends very well. Additionally, depending on the actual behavior of the data trend, window sizes are performing differently. Therefore, we choose an in-between solution of window size of 7 days which seems to give results that generalize fairly well. The slice size does not affect the results too much when it is not chosen too big. The benefit in not choosing the slice size to be 1 hour is that much less windows are generated and that the Flink implementation benefits from that performance wise. Therefore, we choose a slice size of 1 day, meaning that a whole day (24 hourly data points) is removed from and added to the window at each step.

6.1.2 Grid search for step size of gradient descent

In order to get the best fitting model of Wikipedia traffic, we apply grid search for the optimal step size of gradient descent. This is done for each page to ensure convergence of the model. We run 20 iterations of batch gradient descent for each step size and calculate the sum of squared errors of the resulting model. For instance, for the main page the optimal step size is 1.0 as presented in table 2. The values of the grid search range from 5.0 to 0.01. This range guarantees convergence for all pages we tested. The model with the lowest sum of squared errors will be used in the anomaly detector.

6.2 Model presentation on Wikipedia "Main Page"

Figures 4 and 5 show plots of the Wikipedia "Main Page" visits (black) and the fitted model (yellow) for the moving average and multiple linear regression respectively. There are a few interesting things to note here. First of all, the data shows a daily seasonality where there are usually peaks around noon and local minima at night. Secondly, the raw Wikipedia traffic data seems to behave strangely from mid-April to early-July where the page visits suddenly drop dramatically and stay at this level during that time period. One would expect the page visits continue to follow the general trend that can be clearly observed before and after that time period. This first observation about the data brings up some concerns about the data quality, since it might have been a problem of the data collection process. However, we cannot trace back the origin or reason of this unusual behavior in the data.

On the other hand, we can also see the difference in how these two models fit the data. The moving average model fits a rather smooth line through the general trend of the data, whereas the regression model also captures the daily seasonality of ups and downs.

Figures 4 and 6 show the threshold of three times the standard deviation around the fitted model for the Wikipedia "Main Page" (red). The moving average model captures the moving variance which makes it adjust to the time period between mid-April and early-July. This time period exhibits less variation of the model. The linear regression model, on the other hand, does not incorporate an adaptive variance, but makes up for this fact since it better fits the data.

Figures 7 and 8 show the residuals of the two models. That means that the function values of the fitted model are subtracted from the raw data. The residuals are shown in blue and the anomaly threshold in red. Any residual above the anomaly threshold is classified as anomaly.

The moving average model does not detect an anomaly for the "Main Page" during this time period. This can be explained by the fact that we use a two-sided moving average which calculates the trend line by incorporating both data from before and after the point in time where this strange behavior happens. In this way, the model anticipates the deviation from the trend because it remains to be a constant deviation for a longer time period. The model interprets this deviation as a rapidly changing trend and therefore does not classify it as an anomaly.

In fact, the linear regression model does detect this change in trend as anomaly. That is mainly because the regression model does not look ahead in time and only predicts it's fitted function based on previously seen data. It cannot anticipate the deviation from the original trend and therefore the actual value is suddenly different by more than 3 times the standard deviation of the residuals.

The two models give two different, but still two reasonable results. The graph of the "Main Page" does not show any

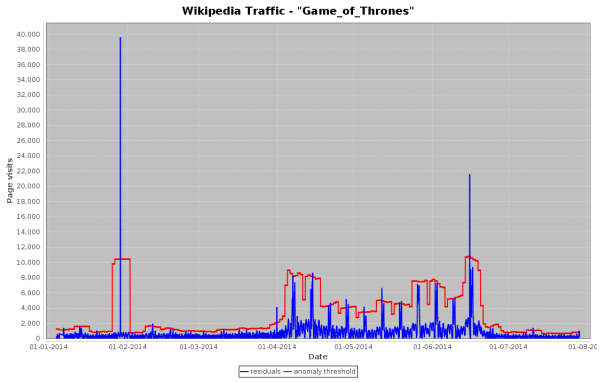

Figure 11: Residuals and anomaly threshold of "Game of Thrones" using moving average

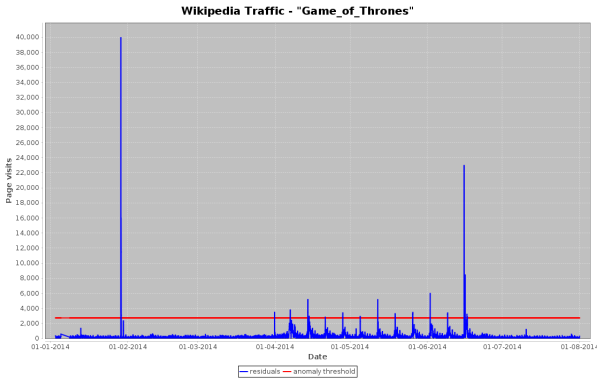

Figure 12: Residuals and anomaly threshold of "Game of Thrones" using regression

temporary peaks on a single point in time which is what we want the moving average algorithm to detect. If the requirement of the model is exactly like that, the moving average model should be preferred. Though, this interesting behavior can still be detected by the linear regression model. So when trends like that are interesting as well, the regression model should be used.

6.3 Trends in politics

We examined some pages which correspond to political trends like Barack Obama, Angela Merkel and Ukraine (related to politics because of the ongoing crisis with Russia). In this paper we show Ukraine as an example since we discovered the most prominent peaks for this page. You can find the charts of the other politics related pages in the appendix.

The crisis in Ukraine began on 21 November 2013. Unfortunately, our data set doesn't cover November 2013, but the crisis heated up in February 2014 when pro-Russian gunmen seized the key buildings in the Crimean capital, Simferopol. This event is also detected by our anomaly detection as the first peak on 28 February. The following peak can be assigned to Russia's parliament approval of President Vladimir Putin's request to use force in Ukraine to protect Russian interests. The third anomaly is aligned with a speech hold by U.S. President Barack Obama who asks Moscow to "move back its troops" on 28 March. The last mayor peak in July was caused by the shot down of a civil airplane (Malaysia Airlines Flight 17) by pro-Russian insurgents. [1]

All the mayor anomalies are detected by the regression model shown in figure 10. But the moving average approach fails to catch the first two anomalies (figure 9).

In general, it is not always as easy as in the case of the Ukraine crisis to detect and assign peaks to real political events. We assume that with an increasing impact on people the detection rate increases as well.

6.4 Trends in Entertainment

When looking at some Wikipedia pages which relate to entertainment in some way, we observe that news or events around people are not extensively covered by the Wikipedia traffic data. Rather recurring events like a TV show are very suitable for anomaly detection in this data setting. Single events as for example the release of a movie or annual events like the Oscars experience exactly that single peak in their data and can easily be detected. We present the example of the English Wikipedia page of "Game of Thrones" as an very interesting example here.

The raw data shows a peak at the end of January and then recurring peaks in a weekly rhythm from the start of April to mid-June. It turns out that the first anomaly corresponds to HBO[4] releasing teasing material about the former upcoming season 4 of Game of Thrones. In fact the weekly recurring anomalies correspond to the airing of each of the episodes of Game of Thrones' season 4. The initial teasing of the TV series actually has the most impact on page visits on Wikipedia followed by the season finale which also shows even more than double the page visits as compared to the other episodes.

Figures 11 and 12 show the residuals and anomaly thresholds for applying the moving average and regression model respectively. We see a good detection of the anomalies for the regression model. The seasonal trend is clearly reduced from the data and the anomaly threshold is reasonable so that all anomalies are detected. The moving average model does not perform that well. It easily detects the biggest anomalies at the end of January and the season finale in mid-June, but it fails to detect 4 of the anomalies produced by the other episodes. In addition, we observe to detect few false positives before the airing of the episodes in the moving average model.

6.5 Trends for Sports

As an example for anomaly detection of sports events we apply our approaches on the FIFA Soccer World Cup in 2014. We use the traffic data of the German version of the corresponding Wikipedia page. The last three peaks represent the quarterfinals, the semifinals and the final. These events are discovered by both models as shown in Figure 13 and 14. Moreover they show a huge anomaly in the end of March. However, this outlier can not be traced back to its origin.

In general, sports events can be tracked more easily because the corresponding pages are mostly updated immediately and are also the first place to browse when people search about sports scores or detailed information about sports events.

[4]Original airing channel of Game of Thrones in the US

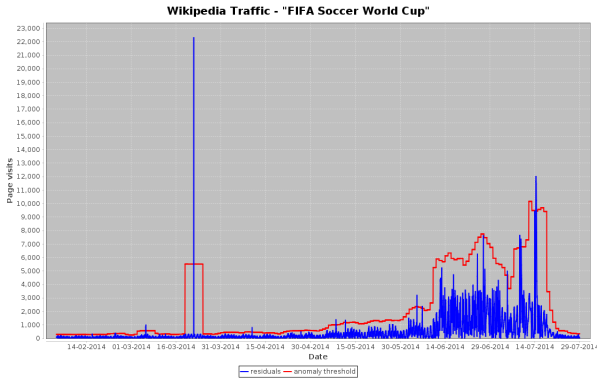

Figure 13: Residuals and anomaly threshold of the FIFA Soccer World Cup using moving average

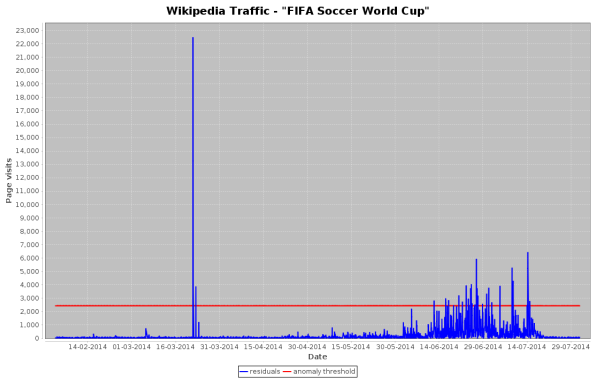

Figure 14: Residuals and anomaly threshold of the FIFA Soccer World Cup using regression

7. CONCLUSIONS & FUTURE WORK

We analyze Wikipedia traffic data using two different models. One being a simple approach of moving averages and one being a more complex approach of multiple linear regression. Our implementations of the algorithms in Flink work on Big Data and analyze the data on a 10-node cluster. Our results show that the regression model generally outperforms the moving average model due to its capacity to fit the daily seasonality found in the traffic data. By removing this seasonality, it detects anomalies using our 3 times standard deviation threshold method more accurately. Many manually validated examples of Wikipedia pages show that the regression model used is reliable. It exhibits less false positives and does not miss on anomalies which the moving average model fails to detect.

However, not every single real world event can be traced solely by the Wikipedia traffic data. The data lacks behind in being representative when aiming for detecting a large variety of real world events. We suppose that people purposefully visit Wikipedia when they can expect to find what they are looking for, e.g. when looking for the latest summary of a TV show episode or the latest result of a sports match. Additionally, it might be used to get background information about news stories when they first appear, e.g. as seen with the Ukraine crisis.

Vice versa, there are also anomalies which undoubtedly are correctly classified because the graphs show an obvious peak at a certain point in time, but we are unable to relate a real world event to it. A reason for this problem might be the rather vast influence of the English Wikipedia site. Since the English language is globally used, there might be regionally specific events which lead to anomalies like that. This makes it harder to track back the exact source of the anomalies when only using a few resources for the news event lookup.

Future work can involve support for streaming data when new raw Wikipedia traffic data is collected and evaluated in real time. One reason we decided on using Apache Flink is its streaming capabilities which enable pipelined stream data processing. That way one could adjust the regression algorithm for classifying incoming data items after training it on previously seen data. The moving average algorithm would have to be adjusted to implement a one-sided moving average so that no future data is needed for calculating the window average and classification.

Another possible extension to the anomaly detection procedure would be to optimize the process of finding real world relationships to discovered anomalies. Instead of manually looking up what happened on a specific date, one could use a news API like the one provided by newslookup.com[5]. This API's news data feed can be combined with a streaming solution to output not only anomalies, but also news events corresponding to the anomalies.

8. ACKNOWLEDGMENTS

We thank all people who helped us with the Flink implementation and problems we encountered. This includes all the help we got from the Flink mailing list.

9. REFERENCES

[1] BBC. Ukraine crisis: Timeline. `http://www.bbc.com/news/world-middle-east-26248275`.

[2] T. Dunning and O. Ertl. Computing extremely accurate quantiles using t-digests.

[3] T. Dunning and E. Friedman. *Practical Machine Learning: A New Look at Anomaly Detection*. O'Reilly Media, 2014.

[4] C. E. Efstathiou. Signal smoothing algorithms. `http://www.chem.uoa.gr/applets/appletsmooth/appl_smooth2.html`.

[5] S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl. Spinning fast iterative data flows. *Proc. VLDB Endow.*, 5(11):1268–1279, July 2012.

[6] R. J. Hyndman. Moving averages. `http://robjhyndman.com/papers/movingaverage.pdf`, 2009.

[7] D. Mituzas. Page view statistics for wikimedia projects. `http://dumps.wikimedia.org/other/pagecounts-raw/`.

[8] Y. Ogata. Statistical model for standard seismicity and detection of anomalies by residual analysis. *Tectonophysics*, 169(1):159–174, 1989.

[9] F. Pukelsheim. The three sigma rule. *The American Statistician*, 48(2):88–91, 1994.

[10] W. G. Schwert. Anomalies and market efficiency. *Handbook of the Economics of Finance*, 1:939–974, 2003.

[5] `http://www.newslookup.com/api_docs.html`

[11] T. Takahashi, R. Tomioka, and K. Yamanishi. Discovering emerging topics in social streams via link anomaly detection. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 1230–1235. IEEE, 2011.

[12] R. A. Yaffee and M. McGee. *Introduction to Time Series Analysis and Forecasting*. Academic Press, 2000.

[13] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 423–438, New York, NY, USA, 2013. ACM.