Dynamische Eventerkennung in Datenströmen

Zhaw School of Engineering

Bachelorarbeit 2019

David Pacassi Torrico, Daniel Milenkovic

May 9, 2019

Contents

1 Abstract

How can events be recognized in a data streams? And what is the definition of an event? Current event recognition procedures define possible events (or event types) statically when the system starts. However, in data streams, the definition should develop dynamically over time as the data and their focus changes over time.

The first goal of this bachelor thesis is to define what an event is and how events can be recognized from static data. Depending on the definition, a suitable dataset has to be found and chosen in order to create a system which is capable of recognizing events. In the second part, the events should be recognized from a data stream and therefor support dynamic event recognition.

This work uses news API's as data streams and tries to assign the received news articles to different events. An event is defined as a list of different news articles writing about the same story. A possible event could be "Brexit" or an upcoming election. It's important to keep in mind that new events can and should be created over time.

Assigning news articles to an event can be done by clustering the news articles. In order to achieve best possible results, a comparison between different clustering techniques has been completed. It was revealed that HDBSCAN is a promising candidate for our use case as it delivered the best results.

Unfortunately HDBSCAN was not made for data coming from data streams but this thesis will show one possible solution on how to deal with this. This work as able to process up to 20'000 news articles with HDBSCAN. A possible continuation of this work could try to extend this number to process more data at once.

2 Introduction

2.1 Problem formulation

What problem do we solve?

2.2 Motivation

Why do we do this? Who might benefit from our work?

2.3 Related papers

Which papers were already published in this regard? Why are they not suitable for our use case?

3 Method

TODO introductionary paragraph

3.1 Test Data

The first important step is to create a test data set to run the evaluations with and verify them.

TODO explain the source and structrue

3.2 Clustering methods

Clustering finds similarities in different news articles based on their content and groups them together, while unrelated news are regarded as noise. The challenge now araises to find an appropriate clustering method, which is able to work with data of varying densities and of high dimensionality.

TODO why hdbscan

3.2.1 HDBSCAN

HDBSCAN is a hierarchical density-based clustering algorithm [**?**]. It extends the well known [insert citation] DBSCAN algorithm and reduces its sensitivity for clusters of varying densities. Another important quality of HDBSCAN is, that it does not need to know the number of clusters up front.

TODO explain some more

3.2.2 K-means

KMeans is a centroid-based clustering algorithm.

TODO explain some more

3.2.3 Preprocessing

3.2.4 Score calculation

The scoring function is essential for measuring the result of a clustering method. The score should tell us how close the resulting clusters are to the ground truth and give us an intuitive understanding about the quality of the clusters. One of the challenges when comparing one clustering with the other, is that the differnet clusters do not share the same label across different clusterings. Which is why scoring functions used for clusterings typically ignore the labels and focus on the shape of a cluster.

Initially we used Normalized Mutual Information (NMI) as our primary scoring function. The NMI calculates the mutual information between two clusterings and normalizes the result by dividing through a generalized mean of the entropy of each clustering. The score proved to work well for our initial evaluations and manual samplings verified the score. Upon running evaluations with higher sample sizes the NMI score started to lose its reliability. An example is given in Table 1, where K-means achieved a rather high score, regardless to the significant difference between the true amount of clusters and the approximation using $\sqrt{n}$.

TODO add number of estimated clusters

To get a more accurate score based on the documents inside a cluster, we developed our own scoring function.

Algorithm	**Sample Size**	**NMI**	$\mid \mathbf{n_{true}} - \mathbf{n_{predicted}} \mid$
k-means	19255	0.754	457
hdbscan	19255	0.742	2

Table 1: K-Means has a higher NMI score than HDBSCAN, while having a much larger difference in number of clusters.

Calculating the score The scoring function first calculates the accuracy for each combination of two clusters, where each clusters belongs to a different clustering. We define the accuracy as

$$\frac{n_{true}}{n_{true} + fp + fn} \tag{1}$$

where n_{true} is the number of elements in the true cluster, fp is the number of false positives in the predicted cluster and fn describes the number of false negatives in the predicted cluster. To illustrate this step with an example, we use T and C as our clusterings, where T is the true clustering and C is the predicted clustering. The clusterings are defined as follows:

$$T = \{\{1,2,3\},\{4,5,6,7\},\{8,9\}\}$$
$$C = \{\{1,2\},\{3,4,5,6\},\{7\},\{8,9\}\}$$

We calculate the accuracy as defined in Equation 1, for each possible pair between T and C starting with $t_1 = \{1,2,3\}$ and $c_1 = \{1,2\}$:

$$\begin{aligned} accuracy(t_1, c_1) &= \frac{n_{true}}{n_{true} + fp + fn} \\ &= \frac{|t_1|}{|t_1| + |c_1 - t_1| + |t_1 - c_1|} \\ &= \frac{3}{3 + |\{\emptyset\}| + |\{3\}|} = \frac{3}{3+0+1} \\ &= \frac{3}{4} = 0.75 \end{aligned}$$

After doing this for each possible pair we get the accuracy matrix A:

$$A = \begin{pmatrix} accuracy(t_1,c_1) & \dots & \dots & accuracy(t_1,c_4) \\ \vdots & \vdots & \vdots & \vdots \\ accuracy(t_3,c_3) & \dots & \dots & accuracy(t_3,c_4) \end{pmatrix} = \begin{pmatrix} 0.75 & 0.375 & 0.427 & 0.375 \\ 0.4 & 0.667 & 0.571 & 0.4 \\ 0.333 & 0.25 & 0.4 & 1.0 \end{pmatrix}$$

As a next step we have to select the most relevant accuracy values from each row of the accuracy matrix.

Finding relevant values in the accuracy matrix non-trivial, since clusters do not share labels across different clusterings. To solve this, we make two assumptions:

1. The higher the accuracy between two clusters, the more likely it is, that both clusters are describing the same group of documents.

2. Each cluster can be associated with a cluster from another clustering only once.

Based on those assumptions we select the highest accuracy value per row, whose column is not already associated with another row. Applying this selection function f to our previously calculated accuracy matrix A results in the set containing the most relevant accuracy values.

$$f(A) = \begin{pmatrix} \mathbf{0.75} & 0.375 & 0.427 & 0.375 \\ 0.4 & \mathbf{0.667} & 0.571 & 0.4 \\ 0.333 & 0.25 & 0.4 & \mathbf{1.0} \end{pmatrix} = \{0.75, 0.667, 1\}$$

As we can see, there were no collisions between columns and we simply get the highest value per row. Consider the following example with an accuracy matrix B, which does contain a collision:

$$f(B) = \begin{pmatrix} \mathbf{0.75} & 0.375 & 0.427 & 0.375 \\ 0.4 & \mathbf{0.667} & 0.571 & 0.8 \\ 0.333 & 0.25 & 0.4 & \mathbf{1.0} \end{pmatrix} = \{0.75, 0.667, 1\}$$

The selected accuracy for the second row is 0.667 instead of 0.8. This is because the fourth column is already associated with the third row, while having an accuracy greater than 0.8. Therefore based on our assumption that clusters cannot be associated twice, the second highest accuracy is used for the second column. In case no association could be found, the value would be set to zero. The full algorithm for the selection process can be found in the appendix as listing 1.

As a final step the average is calculated from the sum of the accuracy with respect to the difference in predicted clusters to the true amount of clusters. In case the predicted amount is lower a normal average will already result in a lower score, since each real cluster without a matching predicted cluster counts as zero. However if there are more predicted clusters than true ones, each true cluster will have a value and the score would appear the same as if there was a perfect prediction in the amount of clusters. To take this into account, the difference is added to the number of true clusters as shown in Equation 2.

$$\frac{s_{accuracy}}{c_{true} + max(0, c_{predicted} - c_{true})} \tag{2}$$

Where $s_{accuracy}$ is the sum of the accuracy values, $c_{predicted}$ is the number of predicted clusters and c_{true} is the number of true clusters. Using our previously selected accuracy values $S = f(A) = \{0.75, 0.667, 1\}$ with $c_{true} = 3$ and $c_{predicted} = 4$, the calculation for the final average would be done as follows:

$$\begin{aligned} score &= \frac{\sum_{i=1}^{|S|} S_i}{c_{true} + max(0, c_{predicted} - c_{true})} \\ &= \frac{0.75 + 0.667 + 1}{3 + max(0, 4 - 3)} = \frac{2.417}{3 + max(0, 1)} \\ &= \frac{2.417}{3 + 1} = \frac{2.417}{4} \\ &= \mathbf{0.604} \end{aligned}$$

The final score for the evaluation of the predicted cluster C with the true cluster is 0.604.

Comparison against NMI After repeating the evaluation shown in Table 1 a second time using the average accuracy per clustering, the score (Table 2) for K-means is much lower than HDBSCAN. This reflects what we would expect based on the big difference in the amount of predicted clusters.

Algorithm	**Sample Size**	**Accuracy**	$\mid \mathbf{n_{true}} - \mathbf{n_{predicted}} \mid$
k-means	19255	0.137	457
hdbscan	19255	0.605	2

Table 2: K-Means has a higher NMI score than HDBSCAN, while having a much larger difference in number of clusters.

The test scenarios in table 3 show the resulting scores of our accuracy score, NMI and completeness. It is important to note that, NMI and completeness work with cluster labels assigned to each document, instead of considing elements inside a single cluster. This means the clustering will be flattened into one dimension, where each document is assigned the label of the cluster it appeares in. The array containing the labels for the first scenario would look as follows: $C = [1, 1, 1, 2, 2, 2, 2, 3, 3]$.

Test scenarios with true clustering $T = \{\{1,2,3\},\{4,5,6,7\},\{8,9\}\}$				
Nr.	Predicted Clustering C	NMI	Completeness	Accuracy
1	$C = \{\{1,2,3\},\{4,5,6,7\},\{8,9\}\}$	1.0	1.0	1.0
2	$C = \{\{1,2\},\{3,4,5,6\},\{7,8,9\}\}$	0.564	0.564	0.694
3	$C = \{\{1,2,3\},\{4,5,6\},\{7\},\{8,9\}\}$	0.895	0.809	0.7
4	$C = \{\{1,2,3\},\{4,5\},\{6,7\},\{8\},\{9\}\}$	0.821	0.697	0.467
5	$C = \{\{1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{7\},\{8\},\{9\}\}$	0.651	0.483	0.204
6	$C = \{\{1,2,3,4,5\},\{6,7,8,9\}\}$	0.434	0.552	0.367
7	$C = \{\{1,2,3,4,5,6,7,8,9\}\}$	0.0	1.0	0.148
8	$C = \{\{7,2,4\},\{8,9,6,3\},\{1,5\}\}$	0.219	0.219	0.524

Table 3: Direct comparison of different scoring functions

TODO explain scores. why is it better than NMI?

3.3 Online Clustering

* time based sliding window * Near duplicate detection for clusters with MinHash and LSH O(log n) * Jaccard Coefficient: number of common elements / (total number of elements - number of common elements) * source https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134 * defining events -¿ new story, changes in story (additions/deletions)

4 Implementation

4.1 Evaluation Framework

The evaluation process is done with our own evaluation framework. The framework allows for automated and repeatable evaluation runs. Results are stored in a database for later analysis. The main features include:

- Defining the number of stories to run the evaluation with and load all news articles from those stories.
- Repeating evaluation runs with different sets of data.
- Providing different vectorizers for converting the textual data into a vector space model.
- Defining a range for each parameter of a clustering method and running it with each possible combination of those parameters.
- Storing the result the result in a database and creating relations between news articles, clusters and evaluation runs. This allows for manual inspection and analysis of individual articles inside a predicted cluster.

The implementation is done with Python. Clustering methods and vectorizers are provided by the Scikit-learn library. We decided to use Scikit-learn because of its rich documentation, the wide range of tools and algorithms it provides for clustering and our previous experience with it. Additionally the framework runs in a fully dockerized environment, which includes the database. This makes it very easy to run locally or on a server.

5 Results

5.1 Evaluation of clustering methods

TODO start with setup and show all parameters to be used

TODO go through the whole process and show best accuracies in the end

TODO include preprocessing

TODO show error bars und use logarithmic scales where necessary

The goal of this evaluation is to find an accurat clustering method for working with news articles. The most suitable algorithm will then be used for the online approach of detecting changes in a news stream based on stories.

To begin the evaluation, we first compare six different clustering methods with each other. K-means is included as a baseline, to put the scores in relation to one of the most commonly used clustering algorithms. As we can see in figure 1, HDBSCAN provides the highest accuracy, while being still being one of the fastest algorithms. Based on this data, we can assume HDBSCAN to be a good candidate to work with further. Therefore we focus on this algorithm for the rest of the evaluation to get more insight into its performance and other important properties.

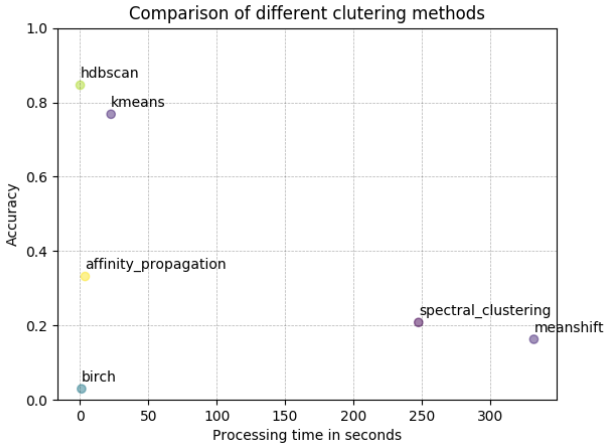

Figure 1: Comparison of different clustering methods with a sample size of approximately 1000 news articles

Increasing the sample size results for both HDBSCAN and K-means in a small loss regarding the accuracy as can be seen in figure 2. However the accuracy seems to stabilize around the 0.7 mark. The drop in the beginning is due to the fact, that for each run, we load the news articles based

While HDBSCAN and K-means provide a similar accuracy, the biggest difference can be noted in the processing time in relation to an increasing number of samples. K-means has a time complexity of $O(n^2)$ in contrast to HDBSCAN with a time complexity of $O(nlog(n))$, which is demonstrated by figure 3. Although running the evaluation has also shown the space complexity for HDBSCAN to be substantially higher for larger amounts of samples than with K-means. Trying to run HDBSCAN with 100'000 news articles caused in a memory error, even with 64GB of RAM, while K-means was able to complete the clustering.

Figure 4 shows, that the difference between predicted over the true number of clusters is fairly low and appears to be roughly linear with the overall number of clusters.

So far we have looked at data provided by the most accurate parameter combination. Figure 5 shows the accuracies of two essential parameters when working with HDBSCAN. The metric is used when

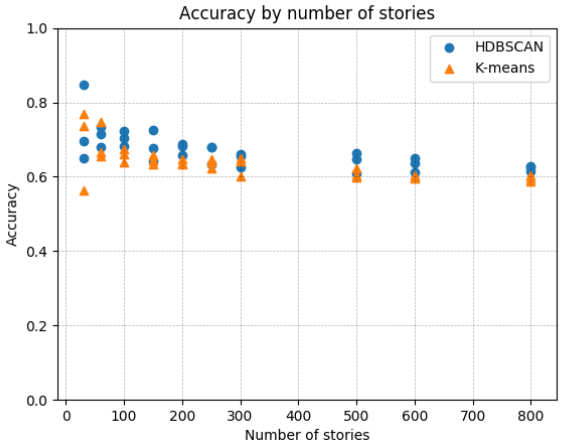

Figure 2: Comparison of the average accuracy between K-means and HDBSCAN

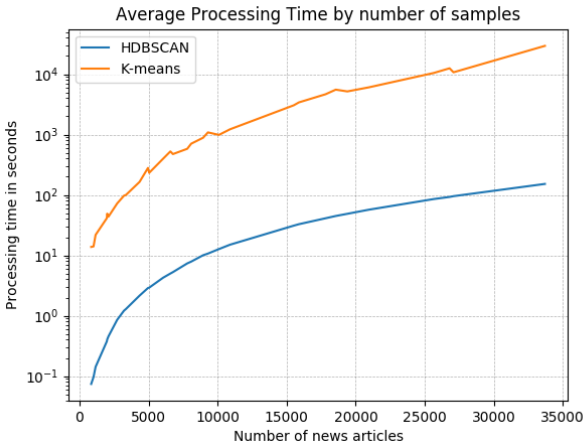

Figure 3: Processing time in seconds

calculating the distances between instances in a feature array. For this evaluation we only consider the metrics *cosine* and *euclidean*, since other available metrics are either unable to work with sparse matrices or the score was too low, while taking significantly longer to calculate. The comparison in figure 5 clearly shows the difference in accuracy between both metrics. Additionally we found that the vectorizer used for the feature matrix, has an big impact on the *euclidean* metric, while the scores using *cosine* were relatively stable. Using the $CountVectorizer$ caused the accuracy to drop significantly in combination with the *euclidean* metric, while using $TFIDF$ gives the accuracies as shown in Figure 5. The second parameter defines the minimum size of a cluster and depends heavily on the dataset. In our case we have clusters with sizes ranging from 1 to 450. We consider clusters with size one as noise, which is why we start with two and evaluate sizes up to nine. The evaluation shows, that a score too low such as $min\_cluster\_size = 2$, will lead to an low accuracy, since the granularity of clustering starts to get too small. Increasing the $min\_cluster\_size$ eventually leads to another drop in accuracy, since we start to loose more clusters the higher the minimum size is set to. The optimum results are achieved with a $min\_cluster\_size$ of 4 or 5.

Being able to handle noise is one of the advantages of HDBSCAN has over the other clustering algorithms. It is reasonable to expect to have a lot of noise in the incoming stream of news articles, once the algorithm is applied in an online setting. However this evaluation is done with a labeled static dataset, containing a minimum amount of noise. The noise ratio shown in Figure 6 is higher, than we

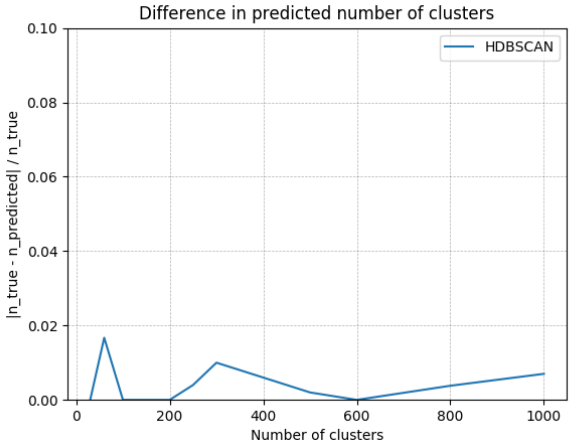

Figure 4: Ratio of difference over predicted with true number of clusters

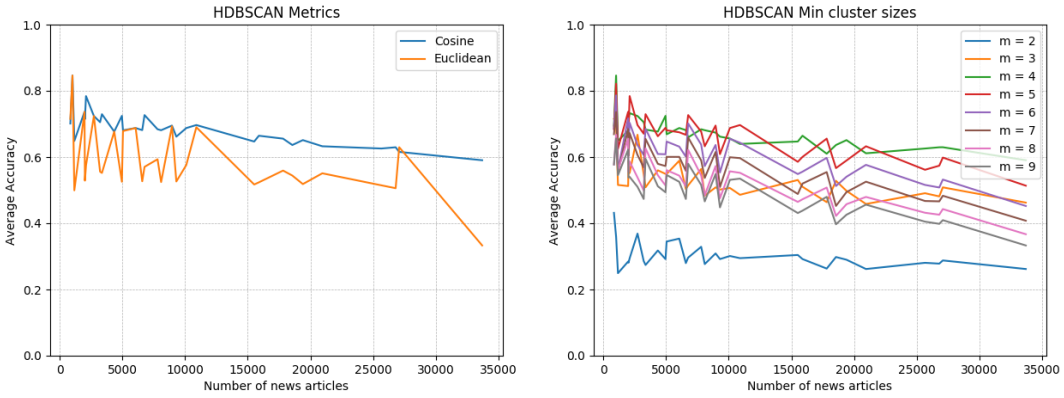

Figure 5: Accuracies for different parameters

would expect from the test data. One reason is that clusters lower than the threshold *min_cluster_size* will be classified as noise. Although considering the optimum accuracy can be obtained by using $min\_cluster\_size = 5$ only approximately 2% of clusters will drop out due to the minimum size. This does explain the noise ratio of 25%. Another possibility is that the data does contain a high number of noisy news articles, but considering the initial substantial effort into cleaning the test data, we estimate the noise ratio to be in the single digit range, rather than anywhere near 20% to 30%.

The evaluation has shown HDBSCAN to be a good candidate to use for news clustering. It provides an better accuracy than K-means, while being much faster to process. The predicted number of clusters is consistent with an increasing number of samples and fairly close the truth. The ideal parameters for our dataset are *cosine* as the distance metric and 4 or 5 as the minimum cluster size. On the flip side the noise ratio is quite high and the space complexity is problematic with larger datasets. Overall HDBSCAN provides a acceptable accuracy, while still leaving room for major improvements.

5.2 Evaluation of online clustering

TODO new topic, topic extended

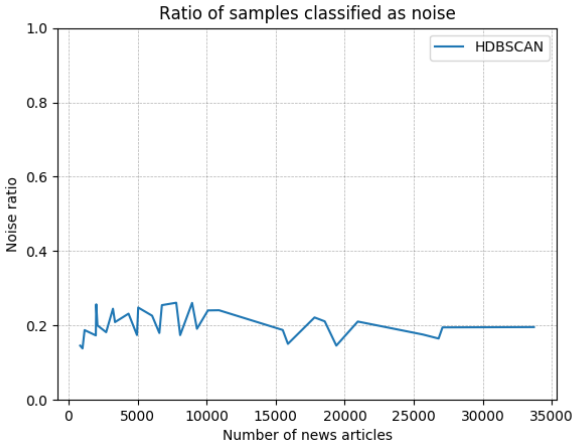

Figure 6: Number of news articles classified as noise.

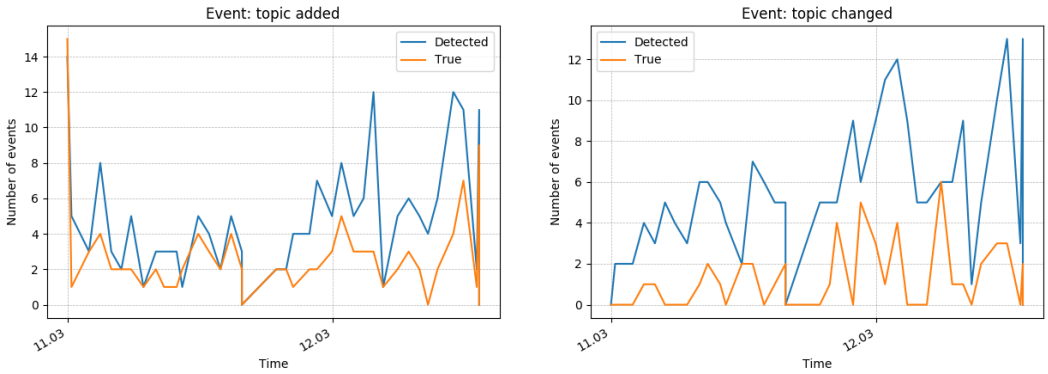

Figure 7: Comparison between detected and true events

Figure 8: Plot work in progress

6 Conclusion

6.1 Summary

Short summary of this work.

6.2 Advantages and Drawbacks

What are the Advantages and Drawbacks of our work?

6.3 Further Improvements

How can this work be improved further?

* improving space complexity and noise ratio of hdbscan * alternatively use hdbscan as an approximation for the number of clusters and a different clustering algorighm for the acutal creation of the clusters.

7 Appendix

7.1 Algorithm for Accuracy Selection

```
    def select_max_values(self, accuracy_matrix):
        unique_indicies = dict()
        row_index = 0
        nrows = len(accuracy_matrix)

        while row_index < nrows:
            ignore_indicies = set()
            max_value_found = False

            while not max_value_found:
                max_value = 0
                column = 0
                for col_index, value in enumerate(accuracy_matrix[row_index]):
                    if value >= max_value and col_index not in ignore_indicies:
                        max_value = value
                        column = col_index

                if (
                    max_value > 0
                    and column in unique_indicies
                    and unique_indicies[column]["row_index"] != row_index
                    and unique_indicies[column]["max_value"] > 0
                ):
                    if unique_indicies[column]["max_value"] < max_value:
                        # The column is already used, but we found a better
                        # candidate. We use the new candidate and set the
                        # cursor to the old one to find a new max value.
                        old_row_index = unique_indicies[column]["row_index"]
                        unique_indicies[column]["row_index"] = row_index
                        row_index = old_row_index
                        unique_indicies[column]["max_value"] = max_value
                        max_value_found = True
                    else:
                        # The column is already used by a better candidate.
                        ignore_indicies.add(column)
                else:
                    # If max_value is greater than 0, we store the value as a
                    # new candiate. Otherwise either the row does not match
                    # any other column or the max_value was low and got
                    # overriden by previous tries and no other match is available.
                    if max_value > 0:
                        # The column is free to use
                        unique_indicies[column] = {
                            "row_index": row_index,
                            "max_value": max_value,
                        }
                    max_value_found = True
                    row_index += 1

        return unique_indicies
```

Listing 1: Select relevant accuracy values from a accuracy matrix.