T. Grust et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2019),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2019 419

Machine Learning Applied to the Clerical Task Management Problem in Master Data Management Systems

Lars Bremer[1], Mariya Chkalova[2], Martin Oberhofer[3]

Abstract: Clerical tasks are created if a duplicate detection algorithm detects some similarity of records but not enough to allow an auto-merge operation. Data stewards review clerical tasks and make a final non-match or match decision. In this paper we evaluate different machine learning algorithms regarding their accuracy to predict the correct action for a clerical task and execute that action automatically if the prediction has sufficient confidence. This approach reduces the amount of work for data stewards by factors of magnitude.

Keywords: IBM[4] Master Data Management, MDM, Machine Learning, Random Forest, XGBoosting, Sorted Neighborhood Method, Data Fusion, Matching, Clerical Task Processing, Duplicate Detection

1 Introduction

Enterprises across all industries have a need to manage master data such as customer, supplier, employee, patient, product or contract information. Master Data Management (MDM) is a discipline comprised of people, processes and technology to manage that master data. Data stewards use automated processes to manage data quality for master data to achieve repeatable, consistent data quality outcomes. One of the key data quality requirements for any MDM solution is the ability to detect and resolve duplicate master data records using matching (matching is also known as duplicate detection). Duplicated master data records exist for at least three reasons. The first one is that before companies introduce an MDM system, master data has been created and managed redundantly and inconsistently in many different application systems. When an MDM system is introduced, master data records from these different sources need to be harmonised and deduplicated to create the 360° view (also referred to as golden record) for a master data entity. The second reason is that in certain industries the sources are outside of the company's control. For example, a healthcare insurance company gets patient data from doctors and hospitals where the sourcing IT systems are outside of the company. Consequently, each time new patient information is received, the matching needs to detect duplicates again. The third reason

[1] IBM Research and Development, Schoenaicherstrasse 220, 71032, Boeblingen, Germany, lbremer@de.ibm.com
[2] IBM Research and Development, Schoenaicherstrasse 220, 71032, Boeblingen, Germany, mariya.chkalova1@ibm.com
[3] IBM Research and Development, Schoenaicherstrasse 220, 71032, Boeblingen, Germany, martino@de.ibm.com
[4] IBM is a trademark of IBM in USA and/or other countries.

doi:10.18420/btw2019-25

is that at the time a new master data record is added to the MDM system, the information might not be complete enough to detect at time of creation that this master data record is a duplicate of some other record and should not be created. Only when a subsequent update adds additional attributes matching might be able to detect that the more complete master data record is now indeed a duplicate to at least one other record and should be merged (also referred in literature as data fusion [BN09]). A typical use case for this is the customer lifecycle. A customer record for a lead is sparsely populated, often times not much more than a name and contact method. As the lead becomes a customer prospect and eventually an active customer additional information like an address, identifiers,etc. are added. Based on this more complete information a match might be found.

For matching, there are two fundamentally different techniques known as deterministic matching and probabilistic matching. Figure 1 shows the basic problem. A base record is compared with pairwise comparison to two other records.

Base Record Indicator	First Name, Middle Name Last Name	Weight	Street, House Number, City, ZIP, Country	Weight	DOB	Weight	Gender	Weight	Identifier	Weight	Total Score
Y	Robert Fisher		Main Street 150 New York 92123, USA		24-10-1975		M		333-22-4444		
N	Bobbie Fisher	3	Main Str. 150 New York 92123, USA	3	24-10-1975	5	F	-0.5	333-22-4444	5	15.5
N	Robert M Fisher	3.3	Main Str. 150 Los Angeles, USA	2.2	24-10-1974	4.5	M	0.5	999-77-8888	-5	5.5

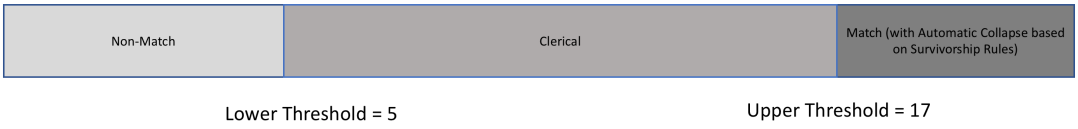

Fig. 1: Matching Example

Deterministic matching uses a set of static rules to decide whether or not two or more records are similar enough and should be merged into a single record. Probabilistic matching typically uses fuzzy techniques comparing the attributes (or set of attributes together, e.g. like the attributes of an address as shown in Figure 1) of two or more records. Fuzzy techniques include for example edit distance, phonetic similarity, nickname resolution, etc. For each attribute (or set of attributes) a weight is assigned how much it contributes for the total similarity score of a record compared to other records to make a non-match or match decision. For example, a gender attribute which has only two values typically contributes with less weight to the overall decision of the matching in comparison to maybe an identifier field as shown in Figure 1. Weights can be positive or negative depending on the similarity found as shown in the Figure 1 with the values in the identifier column. In addition, probabilistic matching considers frequency distribution. For example, in a typical data set of last names in Germany, the last name Cheng is substantially less frequent then Müller. That means if probabilistic matching finds a similarity in the last name field, the base weight for the last name increases in case Cheng is found whereas in the case of Müller it

decreases to reflect the significance of the value frequency. In a given data set with n records, comparing each record with every other record is not efficient for real-time decision making. Techniques like Sorted Neighbourhood Method (SNM) were introduced in [HS95] and various optimisations have been studied in [HS98] and [BN09]. An evaluation of different techniques like traditional blocking, SNM, adaptive SNM, Q-gram, threshold-based canopy clustering, suffix-array based indexing and more has been done in [Ch12] showing variations in performance, completeness and quality. The basic idea is to create a structure acting as a means to quickly find records which share some similarity so that the matching only needs to compare a small subset of records in a small bucket regarding similarity. A well-configured matching algorithm in an MDM system in production aims for buckets of 200 to 500 records at a maximum to enable match decisions with a response time in milliseconds range.

Once the matching algorithm has compared two records the computed weights in probabilistic matching are summarized to a total score which is compared against two thresholds as shown in Figure 1. If the total score is below the lower threshold, the result is considered a non-match and the records are kept separate. If the total score is above the upper threshold, the records are deemed similar enough that based on a set of predefined survivorship rules the records are automatically merged to create a new golden record. If the total score is between the lower and upper threshold a clerical task is created for a human data steward. For the two records shown in Figure 1 which are compared to the base record, the total scores are within the clerical range. A data steward claiming a clerical task is reviewing the records to make a final decision if they should be merged or kept separate.

In two scenarios this approach with clerical processing to manage data quality does not work well. The first scenario is the initial load of the MDM system and and the second scenario is the onboarding of additional sources in sub-sequent deployment phases. In an initial load scenario of medium size with 20,000,000 records with just 1% duplicates in the clerical range 200,000 tasks are created for the data stewards. If a data steward processes 50-200 per day a single data steward would need to work between 1000 to 4000 days until all of them are processed. During that time new clerical tasks are created while the MDM system is used in ongoing mode and other sources are onboarded in a batch load in subsequent deployment phases keeping the task list growing further.

Even if a company hires a data stewardship team to process all these clericals at least several months will pass. This comes at a certain labor cost and not all companies want to invest that much in data quality. We have seen cases where the clerical task list in an MDM system has more than one million pending tasks which is factors of magnitude too large for the data stewardship team to ever catch up and process. In addition to the labor cost consideration, there is also the motivation of the data stewards to be considered. Practice has shown that over time the clerical task processing becomes somewhat repetitive and boring for the data stewards causing frustration with negative impact on performance and hence data quality. If a data steward or a data stewardship team works for a while, a resolution history of clerical tasks is created which contains the matching details and the decision of the data stewards

showing which records have been merged and which ones have been kept separate based on the pairwise comparisons by matching.

The obvious question is why the matching configuration and particularly the thresholds are not adjusted from the beginning so that there are not that many tasks in the clerical range. Companies in certain industries like banking, insurance and healthcare are very reluctant when MDM is introduced to allow a lot of auto-collapse results in fear that there are too many false positives which could cause problems. In banking for example, a false positive caused by an auto-collapse could mean that the related financial accounts are linked to the golden record and someone sees someones else banking details which should not happen. So the upper threshold is set high enough so that not a lot of auto-collapse cases occur. On the other end of the spectrum, enterprises want to avoid issues with too many false negatives which could become problematic in case compliance with anti-money laundering regulations (e.g. like the 4th Anti-Money Laundering Directive [EUAML15]) are required and hence set the lower threshold often times too low. As a result, many match results end up in the clerical range causing the previously outlined issues.

In this paper we want to study how machine learning can be used to reduce the clerical task processing problem.

2 Approach

In this section we describe the approach we have taken to explore how machine learning can be applied to the matching and clerical task processing problem. We applied machine learning to the resolution history of clerical tasks. The basic idea is to train a machine learning model on the resolution history, so that if a new clerical task comes in, the trained model is used to predict the appropriate data steward action. If the trained machine learning model is above a configurable confidence threshold, the predicted action (non-match or auto-collapse) is taken automatically which means the clerical task is not routed to a data steward anymore reducing the workload for a data steward (or the data stewardship team).

Since this was a new area for us, we decided to do an empirical study of different machine learning algorithms to compare them in terms of quality of results as well as runtime characteristics such as performance, memory and CPU consumption. We tested with machine learning algorithms such as Random Forest [Xu17] and [SBP17], XGBoosting, KNeighbors [SBP17], Gradient Boosting, AdaBoost [FS95], Multi-layer Perceptron, Decision Tree, Logistic Regression, Quadrant Discrete Analysis, Gaussian Naive Bayes, Logistic Regression and Support Vector Machines [SBP17].

3 Architecture

The evaluation of machine learning for the use case was not only about finding out if machine learning yields good results - the evaluation also needed to show if a machine

learning component could be added to an MDM system for production use. Figure 2 shows a simplified view of the architecture for an MDM system limited to the areas relevant for this discussion (for a full discussion of all aspects of an MDM system see [Dr08] and [Ob14]). Consumption of an MDM system is through the services layer. The batch and delta-batch processing for initial and delta-loads is typically just a multi-threaded wrapper around the services layer to ensure that for real-time and batch consumption for data consistency reasons the same code is used. The services providing create, update, delete, read and search functionality are used to read and write data to the persistency layer. As part of the create and update services, probabilistic matching is triggered (in the create case that happens all the time, in update cases only if matching relevant attributes are changed). As outcome of matching clerical tasks might be created which data stewards can process through a data stewardship user interface (UI). Every time a clerical task is finished, a new entry to the resolution history is written.

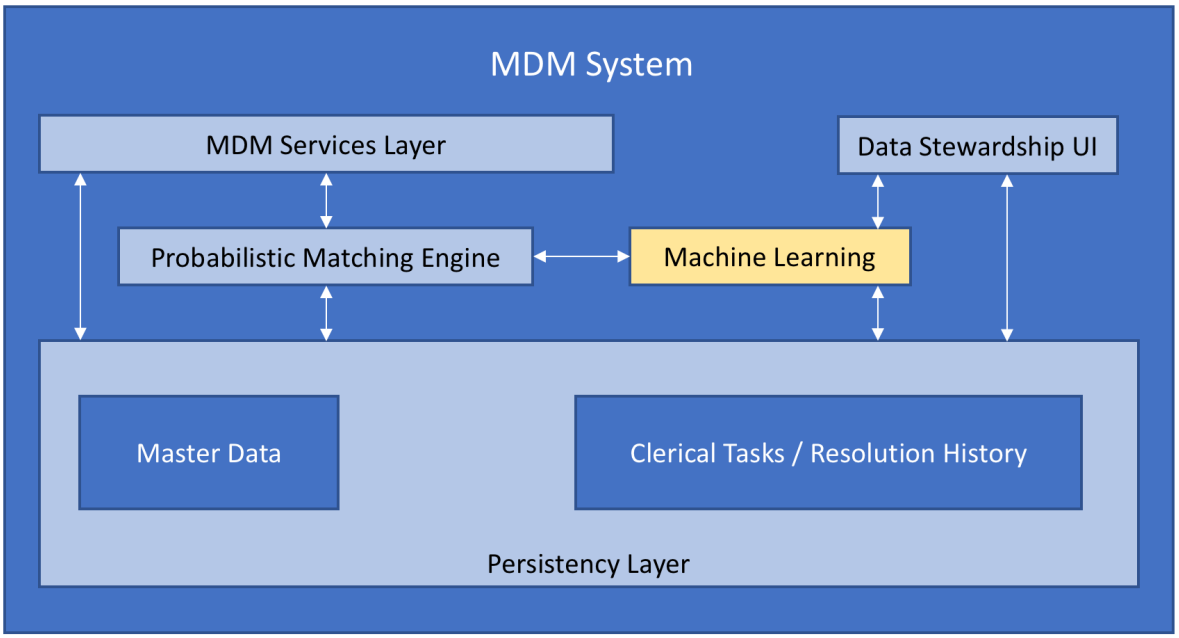

Fig. 2: Target Solution Architecture

Our goal was thus not only to find out if a particular machine learning method is useful for one or both use cases. We wanted to analyze if a trained model can score fast enough to be useful for the first use case of predicting the right action for a new clerical task. MDM systems in production typically have service level agreements against the services layer where the entire service call needs to finish in double-digit milliseconds at most including matching. As a result, the objective for us was if the trained model used for scoring clerical tasks can perform that action in a single to very low double-digit millisecond range at most. Only if that performance target is accomplished the scoring for the prediction can be part of an end-to-end synchronous processing model, otherwise the scoring for the prediction with the auto-resolution need to be asynchronously to the service call. In addition to performance, the scoring needs to scale. In the most demanding operational environments for an MDM

system, the services layer needs to deal with thousands of concurrent calls per second while not compromising on performance. Deploying a trained model for scoring can be done on different technology stacks and we explored some of them to find an answer for the scalability question as well.

4 Implementation

This section describes the implementation and development environment we used to implement different data pre-processing steps as well as to evaluate different machine learning techniques.

We used the Jupyter notebooks in IBM Watson[5] Studio for data pre-processing, training, and evaluation. This allowed a collaborative approach to work on machine learning with multiple developers. As runtime we used Python with the scikit-learn within Watson Studio.

4.1 Training Data

The data we used for training is structured as shown in Table 1. Each line contains one clerical task which is the result of a comparison of two person records by the matching engine using a probabilistic matching algorithm. Each of these clerical tasks contains the numerical results of different attribute comparisons. The training data contains n of these comparison results. The actual number depends on the configuration used by a customer. Each of these values is one feature f_i we are using for machine learning. The higher the value, the higher the likelihood of the compared attributes being the same of the two person records. The label l is the decision ('same' or 'different') the steward took to resolve the clerical task.

l	f_0	f_1	...	f_n
S	2.3	12.9	...	1.2
D	3.1	0.0	...	1.1

Tab. 1: Original Training Data

4.2 Data Pre-Processing

Our training data is retrieved from a clerical task resolution history which contains a very high data quality. There is no need to clean the data before training can be started. However, the clerical task resolution history we are using as training data is skewed. Of all

[5] IBM Watson is a trademark of IBM in USA and/or other countries. IBM Watson Studio documentation can be found here: https://dataplatform.cloud.ibm.com/docs/content/getting-started/overview-ws.html.

decisions around 75% of the tasks where resolved by data stewards with the decision that the two compared user records are the same. Only in the remaining 25% they are considered different.

We evaluated different sampling methods to balance the data. Among these methods were a simple random oversampler as well as four Synthetic Minority Oversampling (SMOTE) methods as described in [Ch02] and [HWM05]. We used different classifiers for the evaluation. The average F1 score and the area under the ROC curve for the sampling methods is shown in Figure 3.

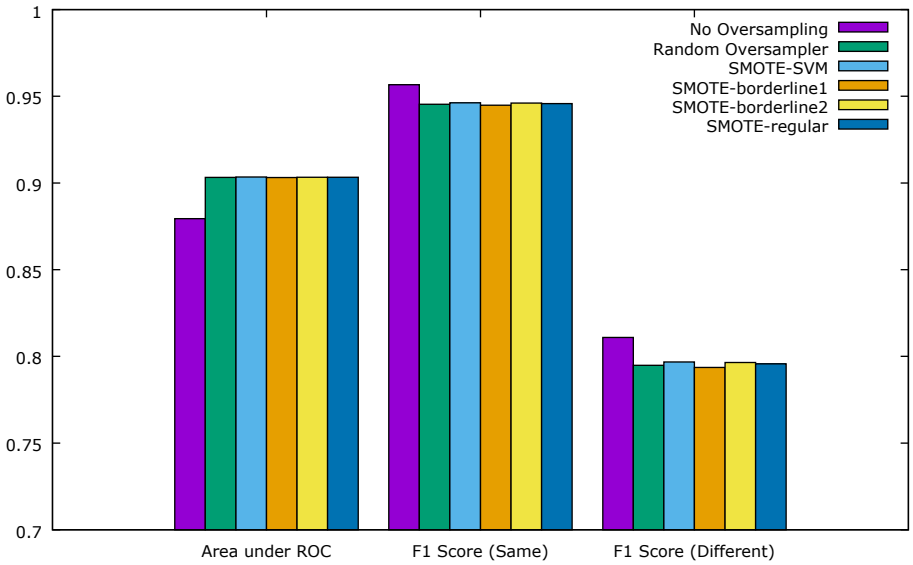

Fig. 3: Sampling Comparison

All sampling methods showed very similar results. They all reduced the F1 score for the minority class. The major reason for this is a reduction of about 10% on average of the precision of the minority class when using oversampling. However, we identified the area under the ROC curve as well as the precision for the majority class (same) increased. This is important for our use case as it is expensive to split user records that were previously merged into one. For that reason, we decided to use random oversampling in the remainder of the evaluations in this paper.

4.3 Performance

For real-time use cases we require high scalability as well as fast response times for predictions. Therefore, we evaluated Spark, Python with Scikit-learn, and MLeap[6] for prediction performance on Random Forest. MLeap allows to serialize Spark, Tensorflow, and Scikit-learn pipelines. The serialized pipelines can then be used for predictions. MLeap

[6] http://mleap-docs.combust.ml/

removes the need for a heavy engine like Spark. The performance of MLeap for predictions and memory consumption reflected this in our tests.

In contrast, Spark showed much higher response times and memory consumption. We reduced this by moving away from running predictions on a Spark cluster but instead in a Spark local instance that is running in the same Java Virtual Machine as our product code. This reduced the response times significantly down to a minimum response time of less than 20ms and a mean of about 40ms. These responses don't change much if executing just one prediction or passing in 1,000 predictions in one batch.

Predictions with Python using Scikit-learn libraries were even faster. A single prediction was returned in about 2ms in our environment. When batching predictions, however, the response time increased using Python to about 15ms for 1000 predictions which is close to the performance we experienced with Spark.

With Python we were able to reach a throughput of 1,500 requests per second each returning a single prediction. These performance evaluations were executed on a single virtual machine with eight 2.4GHz cores and 16GB of memory.

5 Evaluation

This section summarizes the outcome of our comparisons of different machine learning classifiers with pre-processed data as described in Section 4.2. Our comparison includes 10 different classifiers. We compared these classifiers with regards to the area under the ROC curve, accuracy, precision, recall and F1 score.

We started our evaluations with a comparison of ten different classifiers for machine learning to get an overview of their prediction quality. The results of this comparison are depicted in Figure 4. The figure shows the comparison of all ten classifiers with regards to the area under the ROC curve and the F1 score for both labels. In all of these metrics, XGBoosting showed the best results, followed only by a small margin by Random Forest. The difference between XGBoosting and Random Forest is 0.4% for the area under ROC curve and 0.06% and 0.3% for the F1 for the two classes 'same' and 'different' respectively. Other classifiers clearly fall short in comparison.

For a more detailed analysis we moved forward focussing on XGBoosting, Random Forest, and Logistic Regression. We decided on XGBoosting and Random Forest as the best performers in our initial tests. We included Logistic Regression as a reference point as being one of the most widely used classification algorithms.

The probabilistic score we are using for training the models reaches from a negative rational number to a positive. Zero, however, describes the situation when one of the compared entities does not contain the specific attribute that is compared. For example, compare the social security number of two persons. If at least one of the two person records is missing

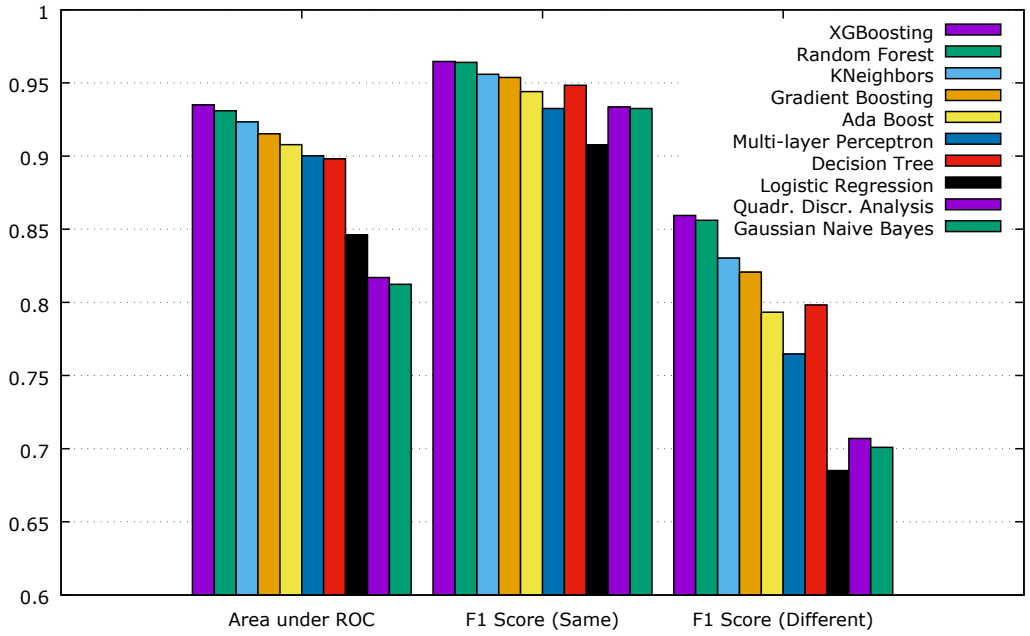

Fig. 4: Classifier Comparison

this number, the comparison value is zero. This means a value of zero does not describe it is more likely than a negative score to be a match or less likely than a positive. Instead, zero values describe that there is insufficient data to define a score for this comparison. To allow different classifiers to compensate for this we introduced a new column f'_n for each feature f_n. This column contains two classes: '0' for when the corresponding matching value equals zero and '1' for non-zero matching values as illustrated in Tables 2 and 3.

f_0	f_1
2.3	1.2
3.1	0.0

Tab. 2: Original Training Data

f_0	f'_0	f_1	f'_1
2.3	1	1.2	1
3.1	1	0.0	0

Tab. 3: Training Data with Artificial Features

Some classifiers trained with these artificial features improved the quality of predictions significantly whereas others showed only very minor improvements as illustrated in Figure 5. The figure compares the F1 score for XGBoosting, Random Forest and Logistic Regression. The quality of predictions of the tree-based classifiers Random Forest and XGBoosting did not change. However, logistic regression improved about 6% for the the majority class 'same' and 11% for the minority class 'different'.

The tree-based classifiers split the branches in a way that they identify the zero values as being different already. This is not possible with logistic regression. For that reason, Logistic Regression benefits from the additional columns. In the remainder of this document all

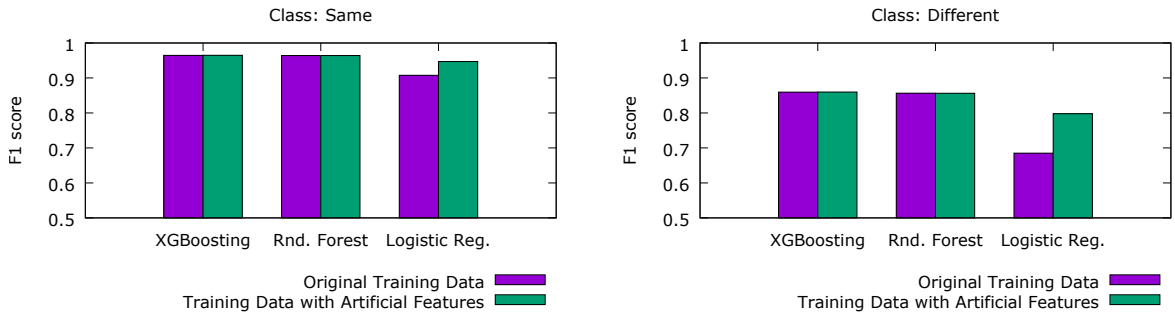

Fig. 5: Evaluation of Artificial Feature Column

results are from classifiers trained with the artificial features. However, for Random Forest and XGBoosting our findings indicate that this is an optional step.

While the quality of predictions is an important metric to compare machine learning classifiers another important aspect is runtime performance.

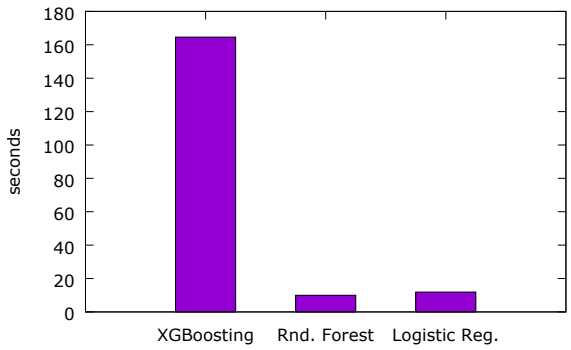

Fig. 6: Training Duration

In the matching use case with classifiers tuned for best prediction quality we observed very different training speeds as shown in Figure 6. Random Forest and Logistic Regression trained a model with about 1 million data records within about 10 seconds using Scikit-Learn libraries. XGBoosting performed much slower. Training the same amount of data takes more than 150 seconds on a single virtual machine with eight 2.4GHz cores and 16GB of memory.

The full comparison of XGBoosting, Random Forest, and Logistic Regression can be found in Figure 7. This shows how similar XGBoosting and Random Forest are for the use case described in this paper.

6 Conclusion

We have shown how historic matching data can be used to make predictions on a steward's matching decisions. We found that with the right machine learning classifier and good input

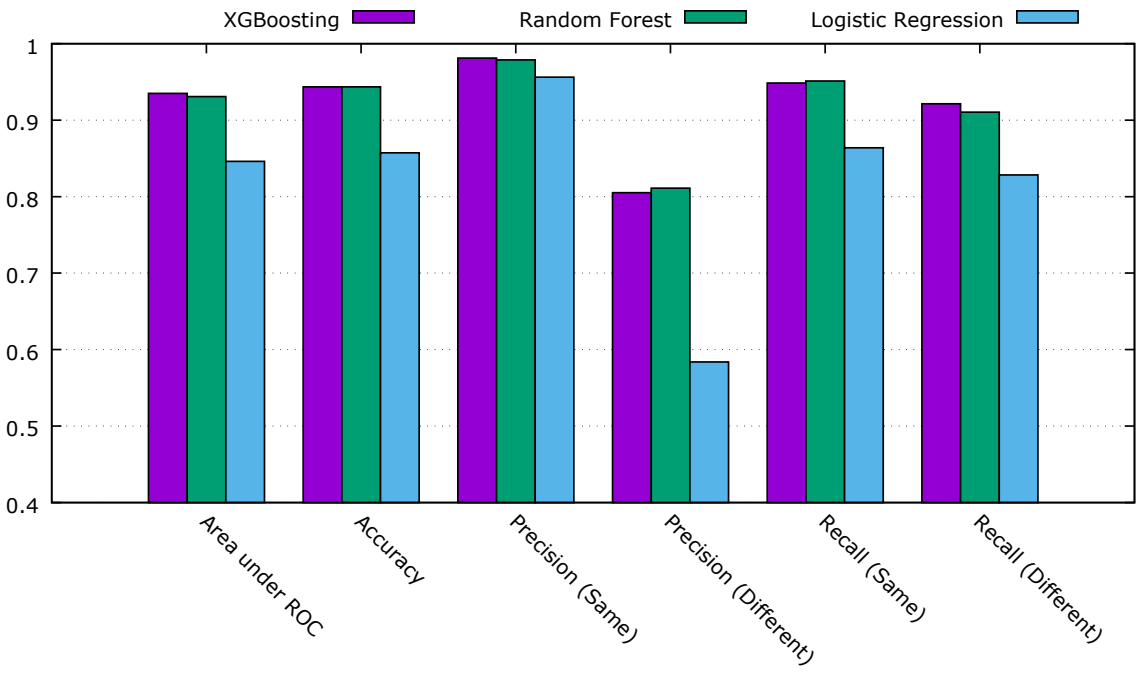

Fig. 7: Full Comparison

data it is possible to predict the steward's decision with an accuracy of over 94%. The precision for identifying two records as being the same can reach up to over 98%. This is particularly important as these predictions lead to a merge of two records. In case of an error it is very costly to divide these records again. Almost all used metrics show results above 90% with the exception precision in the minority class. The precision in this case is only a bit above 80%. Interestingly, this is the only metric in which Random Forest outperforms XGBoosting.

These results were possible after evaluating ten different classifiers and understanding the input data. We realized that artificial columns for zero valued features can be helpful as well as decided to use random oversampling to improve the quality of the predictions.

Finally, we identified Random Forest and XGBoosting as best fitted for the given use case. The prediction results of both classifiers are very similar. However, Random Forest outperformed XGBoosting related to training speed and appears to be the best option for the given scenario.

7 Outlook

One field of further investigations is understanding how the tree-based algorithms prioritize features with regards to their complexity. The score for a name comparison for instance can contain many different individual values because names can be different on many levels

(e.g., spelling, phonetic distance). A birth date, however, can only be different by a certain number of days. We will investigate further how these different distributions of values impact the machine learning classifiers.

A second field for further investigations is to understand how a neural network compares to the classifiers used in this work.

A third field which requires further research is if a probabilistic matching algorithm configuration related to its weigths and thresholds can be optimized through machine learning approach such as logistic regression.

References

[BN09] Bleiholder, J.; Naumann, F.: Data Fusion. In: ACM Computing Surveys (CSUR) Surveys Homepage table of contents archive Volume 41 Issue 1. ACM New York, New York, NY, USA, 2009, URL: http://dx.doi.org/10.1145/1456650.1456651.

[Ch02] Chawla, N. V.; Bowyer, K. W.; Hall, L. O.; Kegelmeyer, W. P.: SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research 16/, pp. 321–357, 2002.

[Ch12] Christen, P.: A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. In: IEEE Transactions on Knowledge and Data Engineering Volume 24 Issue 9. IEEE, pp. 1537–1555, 2012, URL: http://dx.doi.org/10.1109/TKDE.2011.127.

[Dr08] Dreibelbis, A.; Hechler, E.; Milman, I.; Oberhofer, M.; Run, P. v.; Wolfson, D.: Enterprise Master Data Management (Paperback): An SOA Approach to Managing Core Information. IBM Press, 2008, ISBN: 978-0132366250.

[EUAML15] DIRECTIVE (EU) 2015/849 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL, 2015, URL: eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:JOL_2015_141_R_0003&from=ES, visited on: 09/30/2018.

[FS95] Freund, Y.; Schapire, R. E.: A desicion-theoretic generalization of on-line learning and an application to boosting. In: Computational Learning Theory. EuroCOLT 1995. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence), vol 904. Springer, Berlin, Heidelberg, Germany, pp. 20–23, 1995, URL: https://doi.org/10.1007/3-540-59119-2_166.

[HS95] Hernández, M.; Stolfo, S.: The merge/purge problem for large databases. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of data. Pp. 127–138, 1995.

[HS98] Hernández, M.; Stolfo, S.: Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. 2/, pp. 9–37, Jan. 1998.

[HWM05] Han, H.; Wang, W.-Y.; Mao, B.-H.: Borderline-SMOTE: A New Over-sampling Method in Imbalanced Data Sets Learning. In: Proceedings of the 2005 International Conference on Advances in Intelligent Computing - Volume Part I. ICIC'05, Springer-Verlag, Hefei, China, pp. 878–887, 2005, URL: http://dx.doi.org/10.1007/11538059_91.

[Ob14] Oberhofer, M.; Hechler, E.; Milman, I.; Schumacher, S.; Wolfson, D.: Beyond Big Data: Using Social MDM to Drive Deep Customer Insight. IBM Press, 2014, ISBN: 978-0133509809.

[SBP17] Sasikala, B. S.; Biju, V. G.; Prashanth, C. M.: Kappa and accuracy evaluations of machine learning classifiers. In: 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology (RTEICT). Pp. 20–23, 2017.

[Xu17] Xu, Y.: Research and implementation of improved random forest algorithm based on Spark. In: 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA). Pp. 499–503, 2017.