Substation Signal Matching with a Bagged Token Classifier

Qin Wang[1], Sandro Schönborn[2], Yvonne-Anne Pignolet[2], Theo Widmer[3], and Carsten Franke[2]

[1] ETH Zürich, qwang@student.ethz.ch
[2] ABB Corporate Research, {firstname.lastname}@ch.abb.com
[3] ABB Power Grid - Grid Automation, theo.widmer@ch.abb.com

Abstract. Currently, engineers at substation service providers match customer data with the corresponding internally used signal names manually. This paper proposes a machine learning method to automate this process based on substation signal mapping data from a repository of executed projects. To this end, a bagged token classifier is proposed, letting words (tokens) in the customer signal name vote for provider signal names. In our evaluation, the proposed method exhibits better performance in terms of both accuracy and efficiency over standard classifiers.

1 Introduction

Matching utility customer-specified signal names for protection, control and monitoring functions with signal names used by a system provider is a common task in substation automation engineering. To ensure consistency, the system providers maintain an internal library that contains the names to be used for function signals for all projects. This helps the system provider to standardize and streamline its processes and ensures that signal names of important substation automation functions are used in the same manner. On the other hand, the naming schemes used by customers usually differ, both among different customers and compared to provider libraries. Consequently, when starting to work on a new substation automation project, an engineer at the system provider must assign the correct library signal names to customer signal names, a cumbersome, error-prone, and time-consuming process. The matching quality is extremely important to ensure the substation automation systems can work correctly and fits in the customer's environment once deployed and the customer's tools can interoperate with it seamlessly. Hence, in current practice this task is typically carried out by experienced engineers.

The objective of our paper is to find a way to automate this process and thus to improve the engineers' efficiency. More precisely, we present how we devised and evaluated a machine learning-based system that suggest matching library signal names for customer-specified signal names. The system extracts its internal knowledge from past projects that were carried out with a manual signal name assignment. In other words, a repository of past projects is used to build

training and testing data sets for our system. Signal name matching is difficult as customer signal names can be arbitrary and typically contain abbreviations, ambiguity and misspellings. Different naming schemes for lines, e.g., `L1, L2, L3` or `R, Y, B` are both used. In contrast, the provider library consists of a restricted set of known unique and clean signal names. In past project data, the engineers that carried out the matching sometimes made mistakes or ignored the library signal names, in other words the learning data is noisy.

Formal Problem Definition: Since the signal names in the provider library are fixed, this matching problem can be modeled as a text classification task predicting library signal names for customer signal names. Thus, each possible provider signal forms its own class and the problem is an instance of multi-class classification. In our case, we use 3745 possible classes in the provider signal library. The formal task is to *predict* the correct signal class c for a given customer input name $\tilde{x}$, encoded as a string. A model for the prediction is learned from past project data. To support the engineer with multiple possibilities to choose from, the system is to suggest k candidates of matching provider names $c_1, c_2, \ldots, c_k$, sorted by their relevance.

We do not expect the various substation setups to adhere to similar structures and thus consider each name individually. By choosing simple text classification, solely based on customer signal name, we neither require textual similarity between customer and library names nor a common structure among signal names.

Data set: The data set that is used for this work consists of totally 8969 unique pairs of customer signal names with corresponding provider signal names from 170 past projects. Projects have a varying degree of similarity. The overlap across customer signal names is rather low between different projects, indicating the need to standardize names using library signal names.

Method: We propose to use an efficient and accurate token dictionary as a name classifier. In the token dictionary, each word of the customer signal name can vote for possible library names. It is similar to a Naïve Bayes classifier but aggregates and normalizes votes differently. We also explore and adapt a range of existing text classification methods and compare their classification performance as well as their computational efficiency. For this evaluation, we consider standard text classification methods, such as Naïve Bayes, Random Forest and Support Vector Machine and additionally construct a sequence-aware recurrent neural network.

A full version of this paper can be found on https://arxiv.org/abs/1802.04734.

2 System Overview

We setup and evaluate a machine learning pipeline with different classification algorithms to identify library signals from input customer signal names. The pipeline consists of methods for data pre-processing, classification and post-processing. Pre-processing prepares the raw input so it can be processed by the classification algorithms applied afterwards. In particular, they require tokenized

text or numerical vectors as input. In post-processing, we ensure constraints, such as respecting known antonyms in the final signal name, are satisfied.

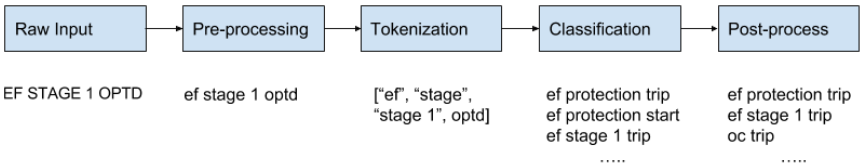

Fig. 1. Proposed pipeline to identify library signals from input customer signal names

Pre-processing Pre-processing consists of normalization, data cleaning and tokenization. Data cleaning is mainly relevant to build a good training set while normalization and tokenization are always applied, also in test scenarios.

Normalization: Signal names are normalized to lower case. This avoids mismatches due to different capitalization methods.

Cleaning: Signal name pairs where the provider name does not occur in the library are removed. These cases exist because either special naming schemes were required by the customer, or the project was implemented before the library was created. Furhtermore, we remove all examples that have identical customer and internal signal name. Theses examples can easily be recognized and predicted, and are thus not our main interest and would not help to discriminate between different methods in the evaluation. After all the normalization and cleaning steps, we have a dataset appropriate for learning and testing.

Tokenization: Most classification methods rely on tokens extracted from signal names. A token is typically a single word in a multi-word signal name. We split between words using a set of separator characters, except numbers nouns. In this case, we tokenize `Noun N` into `Noun` and `Noun N` to capture the context of the number. Empty tokens are discarded. Customer signal names are represented as vectors of token counts, similar to the bag-of-words model [1].

Classification Classification algorithms take the tokenized signal as an input and identify the best matching library signal classes. We evaluate Naïve Bayes, Random Forest, and linear Support Vector Machines (SVM). Furthermore, we present a Neural Network approach and devise a token dictionary algorithm, which turns out to be well suited for this problem in terms of performance and hardware demands. A basic lookup table serves as a comparison baseline.

Post-processing For substation signal mapping, there are antonym token pairs that should not appear simultaneously on customer and internal signal names. For example, if "underfreq" is a token in the customer signal name, then the prediction should never contain "overfreq". In addition, there are also keywords which must appear on both customer and internal sides. For example, if "interlocked" is a token in the customer signal name, then the prediction should also contain this key token. In order to make our predictions more accurate, a post-processing pipeline is implemented to penalize predictions which contain an antonymous token and reward those with the same keywords as customer signal.

This processing step basically reorders our list of predictions to make sure that better predictions are shown on top.

3 Classification Methods

In this section we present the core classification methods we evaluate later. We implement them using the Scikit-Learn [2] and Tensorflow [3] libraries.

Standard Classification Methods

Lookup Table: We use a simple lookup table as our baseline. For each customer signal in the training set, we maintain a list of corresponding library signals. The list is sorted by appearance frequency in the training set. Given a test customer signal, the table returns a sorted list of up to k library signals.

Naïve Bayes: This method assumes conditional independence among multinomial token occurrence probabilities for each class. Despite the simplifying assumption, it often performs well, even with small training sets. The resulting model typically provides fast classification with a moderate memory footprint.

Random Forest: Random forests are ensemble learning methods that counteract single decision tree's shortcomings by taking many trees into account [4]. Contrary to Naïve Bayes classifiers, they can also represent (non-sequential) dependencies among tokens. Random Forests are efficient to train and classify but can require a lot of memory to store all the trees.

Support Vector Machine: Most text classification problems are linearly separable [5] and Random forests are not optimal for very high dimensional sparse feature vectors. Hence linear Support Vector Machine (SVM) classifiers that maximize error margins are widely used for text classification. SVM training is typically memory-inefficient for large datasets and sparse features. We thus resort to the stochastic linear Hinge loss SVM described in [6] to reduce computation time and memory footprint. Splitting part of the training set for probability calibration enables us to suggest k candidate signal names.

Recurrent Neural Network

All methods based on bag-of-words tokens ignore the order of words in a signal name. To capture the sequence of tokens in a signal name, we implement a recurrent neural network of the LSTM-type to classify a sequence of "GloVe" [9] embedding vectors of individual tokens.

Such a setup has been successfully applied to large text classification problems but requires a lot of resources [8].

Vectorization and Embedding: Unlike the above models that use bag-of-word counts as input features, we use pretrained GloVe word embeddings.

Classification network: Bidirectional Long Short-Term Memory (BLSTM) networks have been shown to outperform unidirectional LSTMs, standard Recurrent

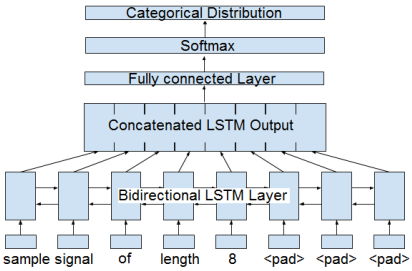

Fig. 2. BLSTM signal classification.

Neural networks, and Multilayer Perceptrons on text tasks due to their stronger ability to capture contextual information [10]. In our model, one BLSTM layer processes the sequence of token vectors as input and generates output vectors. In the cell, we use the standard sigmoid activation function. All output vectors are concatenated and fed to a fully connected classification layer with a softmax output providing the predicted class probabilities.

Optimization We train the network by minimizing the cross-entropy loss of softmax logits using the Adam Optimizer [11] and a learning rate of 10^{-3}.

Token Dictionary

A lookup of the complete signal name, as with the lookup table, is too specific and does not generalize well. But typical customer names still contain specific words which indicate the appropriate library name, almost like keywords. We thus introduce a token dictionary which looks up each token individually and lets it vote for library names it appeared with in the past. Voting allows for ambiguity where the same keyword appears in many classes. Each token votes for all possible hypotheses that could have generated it.

The test signal $\tilde{x}$, to be classified, is treated as a set of its N tokens $\tilde{x} = [t_1, t_2, \ldots, t_N]$. Each token t_i votes for all classes according to the frequency of co-occurrence. The vote of a token t_i for class c is its number of occurrences in examples for said class $n(t_i, c)$. The weight of a vote for class c given token t_i is computed from $n(t_i, c)$, the frequency the token appeared in samples of class c.

$$v(c \mid t_i) = \frac{n(t_i, c)}{\sum_{c'} n(t_i, c')} = \frac{n(t_i, c)}{n(t_i)}. \tag{1}$$

By normalization, the total vote of a single token is split among all possible classes. Common tokens will only contribute weak votes compared to more discriminative tokens. This effect is similar to the one achieved by inverse document frequency normalization. All token votes are summed to form the total vote for for the complete customer name.

$$v(c \mid \tilde{x}) = \sum_{i=1}^{N} v(c \mid t_i). \tag{2}$$

The normalized votes form a probability distribution over all possible K classes.

$$P(c \mid \tilde{x}) = \frac{v(c \mid \tilde{x})}{\sum_{c'=1}^{K} v(c' \mid \tilde{x})}. \tag{3}$$

Formally, such a classifier is a bagged collection of discriminative, weak, single-token classifiers. The token dictionary classifier works as a bag-of-words model. The resulting vote aggregation adds individual contributions and is thus different from multiplying token likelihoods $P(t_i \mid c)$ in the Naïve Bayes classifier. Also, consider the different normalization of token likelihoods and single-token posterior (3). Aggregation of additive votes typically leads to broader prediction distributions than in the Naïve Bayes case. Also, adding votes ensures that a single token can "activate" a library name while all other typical words are

absent. To ensure such behavior, the Naïve Bayes method needs explicit prior initialization, e.g. with Laplace smoothing [7].

By choosing the top k classes with maximal $P(c \mid \tilde{x})$, the algorithm can be easily extended for multiple predictions. It is based on standard Python hashtables, uses little memory and is extremely fast at learning and predicting.

4 Evaluation

All evaluation experiments are run on a workstation with a 4.4.0 Linux kernel, an Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz processor (four cores), and 8GB of RAM memory. Scikit-learn and Tensorflow are the main libraries we used. For the comparison, results of Lookup Table, Naïve Bayes, Random Forest, SVM, LSTM, and our proposed token dictionary are shown here. The models are evaluated on the dataset described in Section 1, including all tokens appearing in the library. 34 (20%) of the projects are used as a (randomly chosen) test set.

Classification performance All testing models outperform our baseline lookup table by a considerable margin. For single-prediction results, random forest outperforms all the other models in term of accuracy, F1, and Recall [7]. In addition, `Top 10 accuracy`, where the top 10 entries of a prediction list for a single query is considered as a match if the true label appears, is provided. Here, the proposed token dictionary and Naïve bayes outperform other models by at least 5% and achieve 91% accuracy. LSTM performs worse than other classifiers despite its large model size, indicating that temporal token dependencies are not crucial in our problem.

Method	Accuracy	Top 10 Accuracy	F1	Recall
Lookup Table	0.66	0.74	0.67	0.66
Naïve Bayes	0.70	**0.91**	0.69	0.70
Linear SVM	0.69	0.90	0.69	0.69
LSTM	0.68	0.85	0.68	0.68
Random Forest	**0.78**	0.88	**0.79**	**0.78**
Token Dict	0.73	**0.91**	0.73	0.73

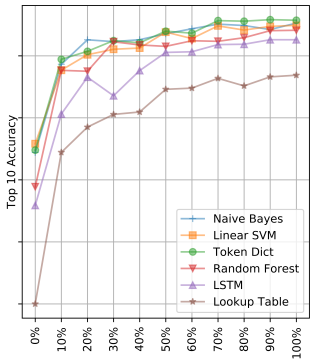

Fig. 3. *Left:* evaluation results on full training set. The best values are printed in bold font. *Right:* top 10 accuracy against amount of training data.

In addition to standard evaluation methods, the influence of the amount of training data on accuracy is measured. As shown in Figure 3, in terms of accuracy, most methods continuously improve when more data are fed. However, this improvement is not significant: less than 5% difference is achieved when when using 100% instead of 50% training data.

Run Time Table 1 presents the runtime measurements of the training and testing phase, using the whole training set for all models. In terms of training

time, LSTM is significantly slower than the others because of its large number of parameters to train. SVM can be trained within ten minutes. The other three models, Naïve Bayes, Random Forest, and Token Dictionary execute the training within less than one minute. In terms of prediction time, all models except SVM respond to each query within 0.1s on average, indicating that these models can be directly used by engineers on a standard workstation without inflicting a bad user-experience. The fastest model, Token Dictionary, processes more than 70 queries within a second, making it the ideal choice in terms of user-experience.

Method	Training time (s)	Mean prediction time (ms)	Peak Memory Usage (MB)	Model Size (MB)
Naïve Bayes	21	17.7	1061	1.2
Linear SVM	361	147.7	1250	1.9
LSTM	12083	34.9	3232	294.1
Random Forest	41	71.9	4583	20.8
Token Dict	16	12.8	143	0.7

Table 1. Runtime and memory consumption.

Memory Consumption The last evaluation concerns memory. Although all models currently fit on the 8GB machine, it is important that the algorithms still work when more data are available in the future. As shown in Figure 1, token dictionary is memory-friendly and consumes less than 150 MB even for the largest training set we have. In comparison, random forest and LSTM model requires 4583MB and 3232MB. These results indicate that random forest and LSTM models might need additional memory on a workstation if more training data are available, while token dictionary is able to capture the mapping relationship between tokens and classes using a rather small amount of memory. Note that the models are compressed.

Discussion In the evaluation of the classifiers under scrutiny we haven seen that the token dictionary features very good classification results combined with favourable running time and memory consumption. A considerable part of the latter is probably also due to the fact that it has been implemented outside the scikit-learn framework. For example one notices a seven-fold difference in the peak memory usage of Naïve Bayes compared to the token dictionary, which cannot be explained by the complexity of the method. Since the classification performance of the token dictionary exceeded the performance of the other methods we did not re-implement the other methods for a more accurate resource consumption comparison. Among further experiments we ran on this data set we evaluated the classification results when expanding abbreviations and observed that it does not bring a significant improvement. On the other hand, including 2-grams and 3-grams (sequences of 2 and 3 tokens) in the token dictionary does improve the classification as inter-token dependencies can be captured with little additional effort. Thanks to the evaluation of the accuracy compared to

the number of training files used, we have seen that our approach (regardless of the classification method) can produce good results already for relatively small data sets. More precisely, even if only 35 past projects are used for training, the predictions offer a high enough accuracy to reduce the engineers' workload.

5 Conclusion

We modeled the substation signal name matching task as a classification problem and evaluated a set of common machine learning methods as well as a bespoke LSTM and token-based dictionary classifier on a data set built from past substation engineering project. Our proposed token dictionary method offers the fastest and most memory-efficient solution for the given task. Moreover, it gives the most accurate list of suggestions and competitive single-prediction results. A potential direction of future work concerns unseen tokens. Due to the nature of bag-of-word models, when encountering new tokens, it is impossible for these classifiers to convert these tokens into features, thus they will fail to capture the information in them, leading to lower-quality predictions. One way to address this is to use a distance measure finding close known tokens to replace it.

References

1. Hotho, A., Nürnberger, A., Paaß, G.: A brief survey of text mining. In: Ldv Forum. Volume 20. (2005)
2. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. Journal of Machine Learning Research 12 (2011)
3. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
4. Liaw, A., Wiener, M., et al.: Classification and regression by randomforest. R news **2**(3) (2002)
5. Joachims, T.: Text categorization with support vector machines: Learning with many relevant features. Machine learning: ECML-98 (1998)
6. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT'2010. Springer (2010)
7. Manning, Christopher D and Raghavan, Prabhakar and Schütze, Hinrich et al.: Introduction to information retrieval. Cambridge university press (2008)
8. Tang, D., Qin, B., Liu, T.: Document modeling with gated recurrent neural network for sentiment classification. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). (2015)
9. Pennington, J., Socher, R., Manning, C.: Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). (2014)
10. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional lstm and other neural network architectures. Neural Networks **18**(5) (2005)
11. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)