A Spectral Method that Worked Well in the SPiCe'16 Competition

Farhana Ferdousi Liza FL207@KENT.AC.UK
Marek Grześ M.GRZES@KENT.AC.UK
School of Computing
University of Kent
Canterbury, Kent, CT2 7NY, UK

Editor:

Abstract

We present methods used in our submission to the Sequence Prediction ChallengE (SPiCe'16)[1]. The two methods used to solve the competition tasks were spectral learning and a count based method. Spectral learning led to better results on most of the problems.

Keywords: Spectral Learning, Rank, Sequence Prediction, Hyperparameters

1. Description of the SPiCe 2016 Competition

The Sequence PredictIction ChallengE (SPiCe) was an on-line competition about guessing the next element in a sequence. Training datasets consist of whole sequences and the aim is to learn a model that allows the ranking of potential next symbols for a given test sequence (prefix), that is, the most likely options for a single next symbol. The evaluation process was interactive: on submission of an answer for a prefix, another prefix was presented and so on. Once rankings for all prefixes were submitted then the score ($NDCG_5$ explained below) of the submission was computed. The score is a ranking metric based on normalised discounted cumulative gain computed from the ranking of 5 potential next symbols starting from the most probable next symbol. Suppose the test set is made of prefixes $y_1, \ldots, y_M$ and the distinct next symbols ranking submitted for y_i is $(\hat{a}_1^i, \ldots, \hat{a}_5^i)$ sorted from more likely to less likely. The target probability distribution of possible next symbols given the prefix y_i, $(p(.|y_i))$, was known to the organisers. Thus, the exact measure for prefix y_i could be computed using the following equation:

$$NDCG_5(\hat{a}_1^i, \ldots, \hat{a}_5^i) = \frac{\sum_{k=1}^{5} \frac{p(\hat{a}_k^i|y_i)}{log_2(k+1)}}{\sum_{k=1}^{5} \frac{p_k}{log_2(k+1)}}$$

where $p_1 \geq p_2 \geq \ldots \geq p_5$ are the top 5 values in the distribution $p(.|y_i)$.

The competition used real-world data from different fields (Natural Language Processing, Biology, Signal Processing, Software Verification, etc.), and some synthetic data. The data description was not available during the competition.

The distributions $p(.|y_i)$ were computed differently depending on whether the data is synthetic (and the model that generated it is available) or real. For synthetic data, the true

1. `http://spice.lif.univ-mrs.fr/data.php`

conditional distribution over the next symbol was used. For real data, where the string y_i is obtained as a prefix of a longer string $y_i ax$, the conditional distribution was computed as follows $p(a^{'}|y) = \delta_{a=a'}$. Note that in this case, $p_1 = 1$ and $p_2 = ... = p_5 = 0$. Thus, when applying this metric to real data, $NDCG_5(\hat{a}_1^i, \ldots, \hat{a}_5^i) = \frac{1}{log_2(j+1)}$, where j is such that $\hat{a}_j = a$ (and $j = \infty$ if a is not in the list of predicted next symbols). The score of a submission was the sum of the scores on each prefix in the test sample, normalised by the number of prefixes in the test set.

2. Our Approach

Spectral learning algorithms (Balle et al., 2014; Hsu et al., 2012) are promising thanks to their theoretical guarantees of consistency. Another advantage is that they require relatively little parameter tuning, where the main parameter is the number of hidden states, n. Our main approach was the method proposed by Balle et al. (2014), and we built our solutions using its implementation described in Denis et al. (2016). We also used an n-gram model with smoothing to compare results on different datasets, and in several cases, the n-gram method was better than spectral learning.

3. Details of the Spectral Method

The approach that gave us the best score on most problems is the algorithm proposed by (Balle et al., 2014). The algorithm follows from a duality result between minimal Weighted Finite Automata (WFA) and factorisation of Hankel matrices (H_f). Weighted Finite Automata are classical finite automata and Hankel Matrix of a function is a bi-infinite matrix. The method provides an efficient algorithm that implements the ideas of the following Lemma to find a rank factorisation of a complete sub-block H of H_f and obtains from it a minimal WFA for a function f which maps strings to real numbers.

Lemma 1 *If $H = PS$ is a rank factorization, then the WFA $A = \langle \alpha_1, \alpha_\infty, \{A_\sigma\} \rangle$ with ${\alpha_1}^\top = h_{\lambda,S}^\top S^+, \alpha_\infty = P^+ h_{P,\lambda}$, and $A_\sigma = P^+ H_\sigma S^+$, is minimal for f.*

3.1 Description of the Algorithm

The purpose of the algorithm is to compute a minimal WFA for a function (f) defined on strings $f : \Sigma^* \to \mathbb{R}$ with finite rank n. Here, Σ be a finite alphabet, where σ is an arbitrary symbol in Σ, and the set of finite strings over Σ is denoted by Σ^*. The algorithm assumes that the $B = (P, S)$ is a complete basis for f, and given that basis along with values of f (in our case the values are frequency of substrings) on a set of strings W, where $W = \{\lambda, \Sigma\}$, as input, the algorithm does a rank factorisation of a sub-block of the Hankel matrix to be able to apply the formulas given in Lemma 1. Particularly, the algorithm assumes that $P\ \Sigma^{'} S \cup P \cup S \subseteq W$. Consequently, the values of f on a set of strings can compute the vectors $h_{\lambda,S}$ and $h_{P,\lambda}$, where λ is an empty string. Thus, the algorithm only requires to compute the rank factorisation of H_λ to be able to apply the formulas given in Lemma 1.

The compact SVD for a matrix gives rank factorisation. The SVD of a $p \times s$ matrix H_λ of ranks n can be written as $H_\lambda = ULV^\top$, where $U \in \mathbb{R}^{p \times n}$ and $V \in \mathbb{R}^{s \times n}$ are orthogonal matrices, and $L \in \mathbb{R}^{n \times n}$ is a diagonal matrix containing the singular values of H_λ. In

the algorithm, this formulation has been written differently as follows $H_\lambda = (H_\lambda V)V^\top$. This can be written because $V^\top V = I$ and $V^+ = V^\top$, when V is orthgonal. With this factorization, the equations in Lemma 1 can be written as ${\alpha_1}^\top = h_{\lambda,S}^\top V$, $\alpha_\infty = (H_\lambda V)^+ h_{p,\lambda}$, and $A_\sigma = (H_\lambda V)^+ H_\sigma V$ and these are used in the learning phase of the algorithm. One of the applications of this algorithm is to determine the probabilities of sequences as follows: $P(x_{1:T}) = (\alpha_1)^\top A_{x_1} \dots A_{x_t} \alpha_\infty$.

In practice, the H and H_σ are not known exactly because the number of finite strings can be infinite and as a result there will be some estimation error. However, it is possible to apply the algorithm on approximate $\hat{H}$ and $\hat{H_\sigma}$. Here, approximation means the $\hat{H}$ and $\hat{H_\sigma}$ can be calculated from a number of strings in W rather than from all strings of W.

3.2 Relevant Parameters

When this algorithm is used for practical purposes with approximate $\hat{H_\lambda}$ and $\hat{H_\sigma}$, for $\sigma \in \Sigma, \hat{h}_{\lambda,S}$, and $\hat{h}_{P,\lambda}$, the algorithm receives as input the number of states n of the target WFA. This is due to the fact that noise can make the rank of $\hat{H_\sigma}$ different from the rank of the H_σ and the algorithm has to ignore some small singular values of $\hat{H_\sigma}$, which correspond to zeros in the original matrix. This is done by computing a truncated SVD of $\hat{H_\lambda}$ up to dimension n. As the zeros in the original matrix do not have much impact on the final result, Bailly (2011) has shown that when empirical Hankel matrices are sufficiently accurate, singular values of $\hat{H}$ can yield accurate estimates of the number of states n in the target. Usually the number of states is chosen based on some sort of cross-validation procedure.

The other important parameter to choose when using the algorithm is the basis. In general there are infinitely many complete basis and for a function of rank n there always exist complete basis with $|P| = |S| = n$. In practice, choosing a basis can be done in the form $P = S = \Sigma^{\leq k}$ for some $k > 0$ (Hsu et al., 2012; Siddiqi et al., 2009). Another approach is to choose a basis that contains the most frequent elements observed in the sample, which can be either strings, prefixes, suffixes, or substrings (Balle et al., 2012). We used the second approach in our method with substrings. Note that the basis vector can be chosen from a subblock of the Hankel matrix where the rows and columns of the Hankel matrix correspond to the substrings and the cells of the Hankel matrix contain the frequencies of the corresponding substrings. In the algorithm, we also choose the length of these substrings. Instead of using substrings, it is also possible to use the prefixes as a row and suffixes as a column. In such a case, the cell of the Hankel matrix can be calculated as the frequencies of the corresponding strings. If the data is informative enough and the frequencies are high enough, the Hankel matrix gives a complete basis without the costly need to look at all possible rows and columns.

3.3 Our Approach to Parameter Tuning

Having the code written by Denis et al. (2016), our main task was to tune the parameters of the algorithm. Specifically, we had to select values for three parameters: the number of states, n, the maximum length of the substring considered in the row of the Hankel matrix, nR, and the maximum length of the substring considered in the column of the Hankel matrix, nC. The objective was to maximise the score ($NDCG_5$) on the problems included in the competition.

We did not apply any of the more sophisticated methods for hyperparameter tuning—such as random search (Bergstra and Bengio, 2012), Bayesian optimisation (Bergstra et al., 2011) or grid-based search (Larochelle et al., 2007)—because the method quickly becomes infeasible when the rank, i.e., the number of states is too high, and we had to stop many experiments manually. Similar to Larochelle et al. (2007), our method included a combination of multi-resolution search, coordinate ascent, and manual search, with a significant utilisation of the last method.

On all problems, our method first initialises nR and nC to 4 and n to 5. Note that the number of rows (columns) in the Hankel matrix is much larger than nR (nC). In the second step, the algorithm starts the process of tuning the number of states n because this was the most important parameter in our preliminary experiments. Random walk is used to select new values of n with the step size being depended on the size of the domain, i.e., the number of observations and the number of sequences. Thus, when nR and nC were kept constant, the value of n was increased or decreased randomly based on the score ($NDCG_5$), i.e., a form of coordinate ascent was performed on n. After the highest score was achieved by tuning n, n was frozen, and the algorithm used the same randomised procedure to tune nR (see Figure 3). Finally, the same procedure was executed to tune the parameter nC (see Figure 6). After tuning nR, and nC, we did not tune n again because, for a given n, a very small improvement was usually observed after tuning nR and nC. On some problems, increasing the values of n, nR and nC to a large number was not possible as the algorithm was becoming intractable.

4. An Alternative Approach: N-gram with Smoothing

We also used 3-gram with Kneser-Ney smoothing (Chen and Goodman, 1996) for the competition datasets.

5. Experimental results

The spectral algorithm led to good scores on problems 1, 2, 3, 9, and 12 (see Figure 1). A common characteristic of all these problems is that they have small numbers of hidden states (low rank). The scores of the predictions made using spectral learning on problems 4, 5, 7, 8, 10, 11, and 13 were not improving with the change of the value of n (see Figure 2), and increasing the size of the basis vector was making the algorithm intractable. On problems 5, 8, and 10, n-gram with smoothing gave slightly better results than the corresponding best spectral solution (see Figure 5). On problem 6, increasing the rank improved the score whereas for problem 11, increasing the rank made the algorithm intractable. On problem 14 and 15, higher rank did not work at all (see Figure 4). The final parameters for the problems can be found in the accompanying website[2].

6. Conclusion

The methods described in this paper placed us in the 4th position with score 9.4082437158 whereas the winning score was 10.4498481750.

2. `https://www.cs.kent.ac.uk/people/staff/mg483/SPiCe2016/`

Acknowledgments

We thank the organising committee for their effort to set up the competition. We also thank Rémi Eyraud for answering our questions related to their software package.

References

Raphael Bailly. Quadratic weighted automata: Spectral algorithm and likelihood maximization. *Journal of Machine Learning Research*, 20:147–162, 2011.

Borja Balle, Ariadna Quattoni, and Xavier Carreras. Local loss optimization in operator models: A new insight into spectral learning. In *ICML*. icml.cc / Omnipress, 2012. URL `http://dblp.uni-trier.de/db/conf/icml/icml2012.html#BalleQC12`.

Borja Balle, Xavier Carreras, Franco M Luque, and Ariadna Quattoni. Spectral learning of weighted automata. *Machine Learning*, 96(1-2):33–63, 2014.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=2188385.2188395`.

James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ACL '96, pages 310–318, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics. doi: 10.3115/981863.981904. URL `http://dx.doi.org/10.3115/981863.981904`.

Arrivault Denis, Benielli Dominique, Denis Fracois, and Eyraud Remi. Sp2learn: A toolbox for the spectral learning of weighted automata. *The 13th International Conference on GI*, 2016.

Daniel Hsu, Sham M Kakade, and Tong Zhang. A spectral algorithm for learning hidden markov models. *Journal of Computer and System Sciences*, 78(5):1460–1480, 2012.

Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proc. of ICML*, pages 473–480. ACM, 2007.

Sajid M. Siddiqi, Byron Boots, and Geoffrey J. Gordon. Reduced-rank hidden markov models. *CoRR*, abs/0910.0902, 2009. URL `http://arxiv.org/abs/0910.0902`.

Appendix A

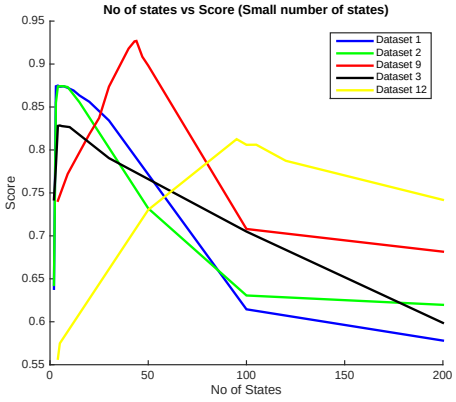

Figure 1: Good Score (SM)

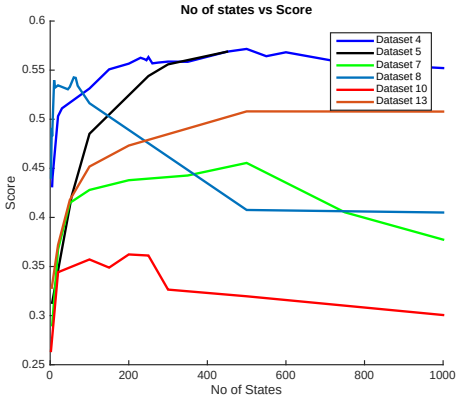

Figure 2: Low Score (SM)

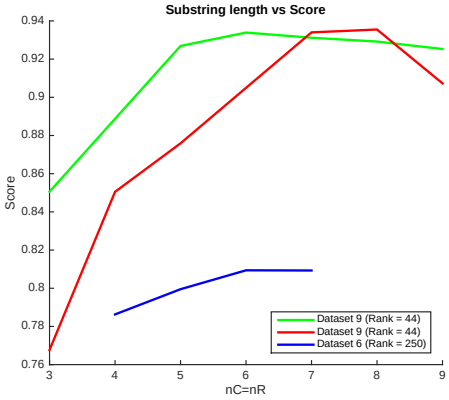

Figure 3: Impact of basis on score

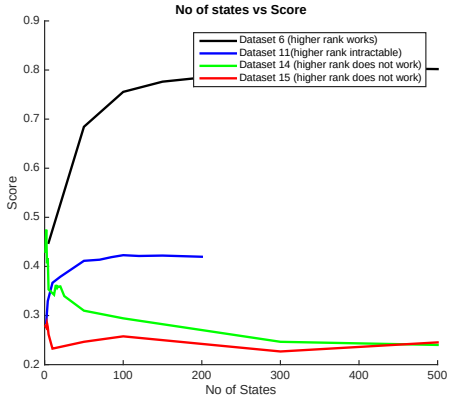

Figure 4: No of states vs Score

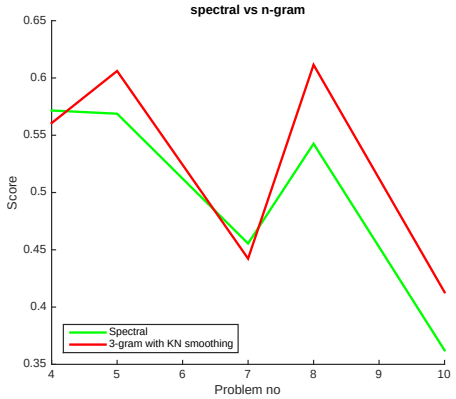

Figure 5: SM vs n-gram

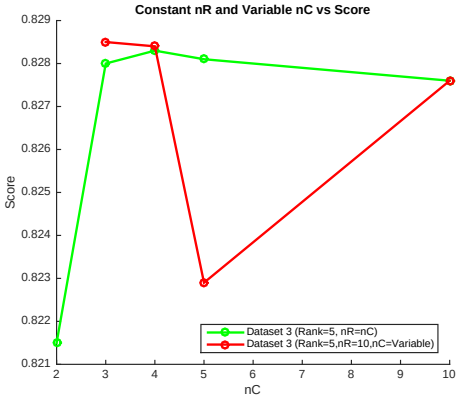

Figure 6: Same and different nR, nC

Appendix B

The code can be found here:
https://www.cs.kent.ac.uk/people/staff/mg483/SPiCe2016/
https://sites.google.com/site/fflgradmission/spice2016

Spectral Method:

To install the python library:
`pip install Sp2Learning`

To execute the code run the following commands:

Usage:
`python spectral_baseline.py train_file prefixes_file output_file`
Example:
`python spectral_baseline.py ../train/0.spice.train ../prefixes/0.spice.prefix.public 0.spice.ranking`

The parameter of the method can be changed in `spectral_baseline.py` code in the following places:

```
#set parameter values
#Estimated rank of the Hankel matrix
rank = 4

#Allow only some of the possible rows and columns of the
matrix
partial = True

#Set max length of elements for rows and column
lrows = 4
lcolumns = 4

#Set which version of the matrix you want to work with
version = "factor"  # "classic" , "prefix", "suffix" ,
"factor"

#Set whether you want to use the sparse or the classic version
of the matrix
sparse = True
```

To get the score run the following:

Usage:
`python score_computation.py rankings_file targets_file`
Example:
`python score_computation.py 0.spice.ranking ../targets/0.spice.target.public`

Table: The final competition parameters and the corresponding competition score. These parameters and scores placed us in the fourth position in the final ranking.

Problem No	Rank	lrows	lcolumns	Score
1	4	5	5	**0.8789916635**
2	6	5	5	**0.8731489778**
3	5	10	3	**0.8248148561**
4	500	5	5	**0.5272911191**
5	3-gram with Kneser–Ney smoothing			**0.6142422557**
6	300	6	7	**0.8096061349**
7	500	4	4	**0.4474728703**
8	3-gram with Kneser–Ney smoothing			**0.6235375404**
9	57	8	7	**0.9324635267**
10	3-gram with Kneser–Ney smoothing			**0.3965168893**
11	100	5	5	**0.4147772193**
12	95	4	4	**0.8113699555**
13	500	5	5	**0.4990697801**
14	2	10	10	**0.4649848044**
15	3	6	6	**0.2899561226**

3_gram with Smoothing:

To run the smoothing code:

Usage:
`python KN_3gram.py train_file prefixes_file output_file`
Example:
`python KN_3gram.py ../train/0.spice.train ../prefixes/0.spice.prefix.public 0.spice.ranking`

To get the score run the following:

Usage:
`python score_computation.py rankings_file targets_file`
Example:
`python score_computation.py 0.spice.ranking ../targets/0.spice.target.public`

Appendix C

A detailed description of our parameter tuning procedure.

```
Parameter_tuning(n,nC,nR)
{
      // step 1
      {
         1. Initialize n=5 , nC=4 and nR=4
         2. Score = blackboxSL(n,nC,nR)
               a. If the computation time is too large (more than 5 hours),
                  then abort and go to step 1 and re-initialize n , nC and
                  nR with smaller values.
         3. Check the score with the online ranking and compare with the
            highest score to check how far the achieved score is from the
            optimum score
         4. Store the score for the next phase. Last_score = score
      }

      // step 2: Initialize n=n+inc , nC=4 and nR=4, here inc depends on
      the training data size and the particular time step

      do{
         1. Score = blackboxSL(n,nC,nR)
               a. If the computation time is too large (more than 5 hours),
                  then abort and go to step 1 and re-initialize n , nC and
                  nR with smaller values.
         2. Check the score with the online ranking and compare with
            last_score to check the direction of the optimum value search
         3. If the score is smaller than the last_score, then set n = n -
            inc.
         4. If the score is larger than the last_score, then set n = n +
            inc.
         5. Store the score and parameter value for the best score.
      }While(not_satisfied_with_result)

      // step 3: once best n is achieved then start tuning nC and nR,

      do{
         1. First set nC= nC + inc1, nR = nR + inc2, where inc1, inc2 >=0
         2. If the computation finish in reasonable time keep increasing
            the value simultaneously or individually (keeping one constant
            while changing other)
      }while(not_satisfied_with_result)

}
```