Form Dialog Progress Prediction

Niels Backer Jouke van der Maas Jacob Verdegaal

Academic Mentor: Assoc Prof Bert Bredeweg, UvA
Client: Jeroen van der Grondelle, Be Informed

Abstract

This paper is about progress prediction in rule-based dynamic forms whereof the total amount of questions is not known beforehand. A baseline predictor is implemented that uses the average time per question, and the maximal remaining question. In this paper possible improvements are tested. The Average Tree, Neural Networks and, Hidden Markov Model methods all outperformed the baseline.

Second year project - BSc Artificial Intelligence, University of Amsterdam
In cooperation with **Be Informed**
June 2013

Contents

1 Introduction

Users expect modern applications to be highly individualized and efficient. In the case of dialogs or forms, they do not want to answer irrelevant questions. Users have very different backgrounds and expertise levels. It is important that user interfaces keep up in innovation with such complex and individualized products. In these contexts, it becomes hard to provide accurate progress prediction, a very important problem from a usability point of view. In addition, traditional measures, such as remaining steps, are not usable in these contexts due to the individualized content.

1.1 Be Informed

"Business processes have conventionally been designed to a large degree with the old hierarchical perspective in mind."[5] In this perspective the flow of a process is core in automatization. In order to automate processes, rules need to be in place to define each flow. Be Informed believes that companies can benefit from a declarative way of defining constraints of processes that are often implicit in old perspective rules. The constraints can then be dynamically used in varying changing processes. When new processes come to existence, only their constraints need to be declared to automate them.

Be Informed provides a way to create semantic networks to declare the constraints processes have. These constraints can now be dynamically used in applications that use this knowledge to complete or support processes. For example, some clients of Be Informed need their users to fill in forms. These users should be automatically classified when done filling out the form, based on the knowledge in the semantic network that Be Informed built.

1.2 Problem Specification

While filling out these forms, which are in core rule driven dialogs, Be Informed wants the clients' users to experience good service and inform them about the remaining time it will take to complete a form. Be Informed was looking for AI techniques that can give an accurate prediction of the remaining time in the dialogs.

The dialogs are truly dynamic due to their dependency on the knowledge base, which is created based on the business rules. It is asking different questions in varying orders to different people every time. This makes progress prediction, an important part of providing a satisfactory user experience, hard, because there is no fixed number of questions or existence of an average time with low deviation.

The application wherein these forms are being filled out in already provides a prediction of remaining time. This prediction is based on the time a user spends on average on answering a question, and the maximal number of remaining questions. But this technique does not work optimally with the declarative knowledge captured in the knowledge network. At the start of a form the system knows what it needs to know about the user to classify him or her, the constraints of the classification process. The application with the knowledge network as spine starts with a question, gets an answer and checks whether it knows enough to classify the user. If not, another question is asked to get more information. This procedure is repeated until the system can classify the user. Note that the system does not know in advance which questions are going to be asked because this depends on answers given.

In AI, techniques are developed to learn from old data. This paper explores the possibilities of usage of these machine learning algorithms to yield better results then the prediction engine in place. Since the problem is about predicting and not classifying it is not a regular machine learning problem. Predicting can be seen as classification, but then with partial data.

1.3 Proposed Product

This project does not aim to find the optimal solution for the prediction problem, but for a significant improvement of the baseline implementation that is currently in use. The development has been guided by the following use cases.

1. *Flexibility*
   Be Informed wants to implement a new algorithm themselves, or replace an old one with a newer version. This should be done without having to update any code, other than the instantiation of this new predictor (including aggregators).
2. *Continuous training*
   Be Informed trains the model on some initial data. After a period of use time, new data has become available. Be Informed wants to add the new data to the model without undoing previous work.
3. *Confidence values*
   Be Informed wants to know how confident the predictor is in its predictions, and use this value to change the information that is displayed to their users.
4. *Aggregation*
   Be Informed doesn't want to be limited to one model; they want the code to automatically pick the best model from a list, or to combine the return values of multiple algorithms. The way of doing this can vary from simple to very advanced.

5. *Serialization*
   Be Informed trains the model. Now they want to close the application and re-open it without losing the work already done.

6. *Concurrency*
   Be Informed wants to continuously train the model as new data comes in, while at the same time using it for predictions.

Use case 1 was realized by separating the development of the predictors from the interface that is used by API consumers. This makes it easy to swap out algorithms or to add new ones. See section 3.1 for more details on this.

Use case 2 was realized by conducting a literature study for a list of algorithms (the long list, see appendix A) that could possibly solve the problem. Here the main priority was the use case, but it was not handled as a hard restriction. Section 2 describes which algorithms were selected and why. This list was used to decide on a smaller set of algorithms to implement during the project.

Use case 3 is of much importance for Be Informed. Since a predictor could be performing excellent on average, it could fail on special cases, which could also occur. Be Informed does not want to communicate incorrect predictions, so each algorithm must have a way of communicating its confidence in its prediction. With the confidence metric predictors could be compared and could be selected on being most user friendly and realizing use case 4. For example: when a predictor has high precision (assumed to be more salable) but low confidence, Be Informed rather communicates a vaguer prediction which is more confident to be correct.

Use cases 5 and 6 are of less importance from a research perspective and can be realized independently.

2 Prediction Techniques

In order to assess the options for progress prediction algorithms, a list of requirements was constructed. All algorithms on this longlist were ranked on their ability to fulfill these requirements, so the most feasible and appropriate ones could be picked.

2.1 Requirements

The algorithms on the longlist were ranked on the following list of requirements, which was put together using the requirements our client and us imposed.

- Algorithms need to provide a confidence measure for their predictions. In order to rank the predictions our algorithms give us, we have to know how confident the algorithms is about the estimation.
- Models must be able to be trained/updated on new examples. Without this requirement, a small change in the model could make all the training up to this point obsolete. Also, further training enhances the models used, and at the starting stage of this project, not a lot of data was available.
- The faster algorithms learn, the better. This means aiming for implementations that use as little data as needed for reasonable accuracy.
- Algorithms should ideally be able to handle two valued features, namely a timestamp and an answer.
- Prediction needs to be fast. The client aims to deploy the predictor interface in a real life situation, and waiting for a prediction for more than a few seconds is unacceptable.
- Smaller models are better. A large model might be precise, but the interface shouldn't be storing large models, because of slower runtime and storage issues.

2.2 Chosen algorithms

The long list, along with the requirement evaluation, can be found in appendix A. From the long list, a short list of methods was selected that best fit these requirements. The justification for choice of methods follows below.

2.2.1 Average Tree

The structure of this algorithm was proposed by Maarten van Someren as an alternative for the more intricate random forest approach. The average tree approach using shrinkage [2] is simple, reliable, and fast. It doesn't take a long time to train or predict, and a confidence measure can easily be acquired by taking into account the amount of different underlying paths.

2.2.2 Back propagation

Using a neural network with back propagation training has several advantages. Next to the fact that prediction is typically very fast ([11]) because it basically consists of a few matrix multiplication operations, neural networks always find an answer, even when encountering unseen data. The latter might seem counter-intuitive to count as an advantage, but it is most definitely preferred over the output an unsmoothed tree-based model would give in this situation, which is none. Also, the output in the highest bucket, in combination with the other bucket outputs and a trace progress indicator, should provide a solid base for measuring confidence.

2.2.3 Hidden Markov Model with Baum-Welch training

This method has proven [7] to be a very reliable way to find the most probable path in comparable situations to the one discussed in this paper. In [7] the problem consist of finding the most probable tasks that must be completed to complete a business process. Here tasks can vary, and tasks take differing times to complete. It was compared to multivariate regression and performed better. Due to the similarity and the variation on the other chosen algorithms this technique was chosen to be tested.

2.3 Disregarded algorithms

2.3.1 Markov Model with Viterbi

Using discrete Markov chains (a type of Markov model), [12] proved to produce an effective estimation model of a similar situation. However, adjusting the problem at hand to fit the Viterbi algorithm, which is an algorithm to find the most likely sequence of hidden states, turned out to be too time-consuming to correctly implement during the available time.

2.3.2 Naive Bayes Classifier

Though this is a solid and fast ([10]) way to do estimation of tree-based structures, it resembles the Markov model, random forest and average tree approaches. They are so much alike that the Naive Bayes Classifier was not picked to be implemented, simply because exploring different estimation techniques was preferred.

2.3.3 Relevance Vector Machine

Relevance Vector Machine (RVM) implementation was discontinued due to the similarity with the back propagation approach, and the fact that RVMs have a tendency to get stuck at a local minimum because of their method of training. This would mean that, even after sufficient amounts of training, the RVM can still form a sub-optimal solution.

2.3.4 Random Forest

The random forest algorithm shows a lot of promise (see [1]), but because of its complexity compared to the average tree (AT) approach, AT was preferred over random forest. Also, random forest might not be able to continuously train, so a full retrain would be necessary to keep it up to date. When working with large datasets, this is a major drawback.

3 Implementation

The implementation of the research system is outlined in the following section (3.1). Then a detailed description of the workings and implementation of the short-list algorithms is given, with the goal of being maintainable if this document is included.

3.1 Architecture

The layout of our interface architecture can be found in appendix B. Due to our desire to be able to implement, train and use multiple algorithms in the same interface, our focus for designing the architecture was modularity and re–usability.

The main interaction with the implemented algorithms happens through the `Predictor` interface. All predictors implement this interface, to provide a common API. As the users answer questions, the application tracks traces. The predictors train on these traces to build their models. After the predictors have been sufficiently trained, the application can ask them for predictions for current users (see figure 1).

Because all predictors use a common interface, it is trivial to swap them out or implement aggregators. Aggregators would implement the same interface, but instead of implementing some algorithm directly, they would proxy the data to the underlying predictors and combine their results.

3.2 Baseline

The baseline looks at historical logs of the form to determine the maximal number of questions (N_{max}). Prediction of remaining questions works by counting how many questions already have been answered ($N_{answered}$) and then calculating the prediction $Baseline_{Steps} = N_{max} - N_{answered}$. For the time prediction the average time per question so far ($T_{average}$) is calculated, the time prediction is given by $Baseline_{Time} = Baseline_{Steps} \times T_{average}$.

3.3 Neural Networks

To implement a predictor based on neural networks, an external library ([4]) was used. Since neural networks require an input and output vector, the data had to be transformed into this form.

The input vector has been defined to contain a normalized value representing the user's answer to the questions. Each dimension in this vector corresponds to a fixed question. For questions that have not (yet) been answered, the vector contains the value 0. Questions that have been answered contain some value between 1 and 2, based

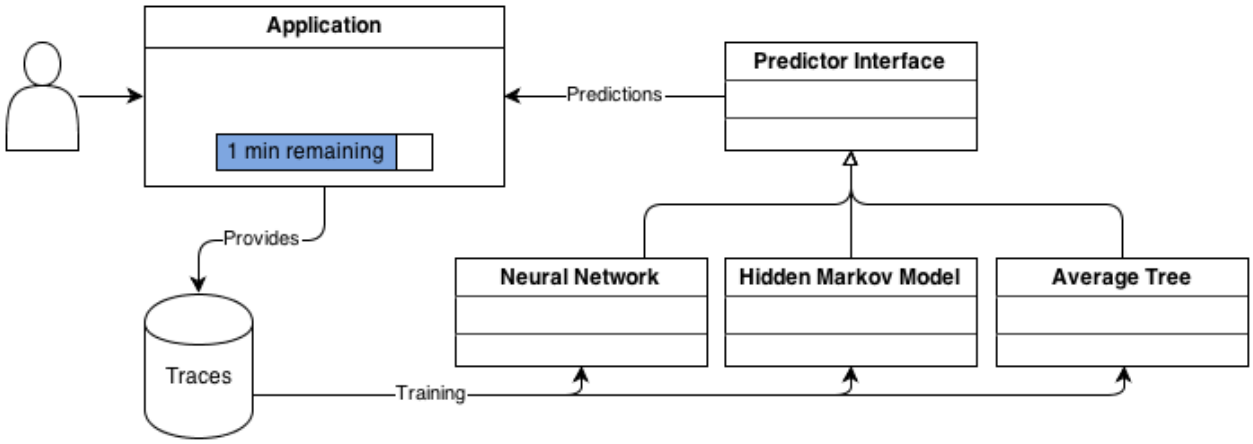

Figure 1: Place of the predictors in the application

on the answers that have been found. E.g. if a question can be answered with **a**, **b** or **c**, the possible values of the corresponding dimension in the input vector (when the question has been answered) will be **1**, **1.5** and **2** respectively. Numeric answers are parsed as such and also normalized to a value between 1 and 2.

To build the output vector, the data is distributed over buckets corresponding to the total duration or amount of steps in the traces. The number of buckets is a parameter to the algorithm. The output vector (for training) contains 1 in the dimension corresponding to the correct bucket, and 0 everywhere else. Predictions of the algorithm consist of a vector of the same size that has values between 1 and 0 in each bucket's dimension. The higher these values are, the more certain the algorithm is that the corresponding bucket is correct. The values cannot be interpreted as probabilities, however, as they do not sum to 1. The confidence value that comes along with predictions is based on these values ($\frac{d_{highest}}{\sum d \in \mathbf{o}}$ for output vector $\mathbf{o}$ with dimensions d.

Training occurs on each complete trace and all of its subtraces (e.g. a–b–c–d has subtraces a–b–c, a–b and a). The algorithm is trained on how much time is left for each trace (i.e. the output vector's bucket corresponds to remaining time), to ensure that partial traces are trained on different values than complete traces. The algorithm continues until an error of less than e is achieved, where e is a value between 0 and 1. When e gets higher, the algorithm becomes less accurate but faster to train. e is a parameter to the algorithm.

3.4 Average Tree

The average tree is a simple algorithm designed for the task that uses the average remaining time or steps at each point in the form to predict the currently remaining time. Each possible path through the questions is indexed. For each path the frequency and average remaining time at that point are saved. It is assumed that more data will lead to more accurate averages. This is why two sets of paths are kept: one containing only the questions (e.g. a–b–c–d) and one containing the questions as well as the answers (e.g. a=1–b=2–c=2–d=1). Paths containing only the questions will be seen more often (if a question has multiple answers), but paths containing both questions and answers contain more information. The prediction consists of a weighted average of the average remaining time for the current path with answers, without answers and of the parent node. The parent node is included because it is likely to have a higher frequency.

To store the paths, a hash table was used in combination with a hashable type. This type stores the path in parts for performance reasons (it can easily recover the parent path). The paths are stored in a hash table, where the values contain the frequency and average remaining time at the corresponding key path. This ensures that predictions can be made in constant time. During training, each complete path is split into its subpaths, which are all added to the hash table separately. Because of this, looking up a child path is as fast as looking up a parent. Two separate hash tables are kept for paths with answers and without. This means that the model size grows with the number of unique answers to each question. The algorithm scales poorly in this regard, but the size should approach some constant for reasonable forms.

The confidence value that is returned with the prediction is based on three values:

1. The progress in the dialog:

$$P = \frac{\textit{Questions so far}}{\textit{Questions in longest path}}$$

2. The amount of data in the model:

$$D_t = \frac{1}{O(\textit{total number of traces})}$$

3. The amount of data in the current node:

$$D_n = \frac{1}{O(\textit{frequency of path})}$$

So the confidence $c = 0.2P + 0.3(1 - D_t) + 0.5(1 - D_n)$ is the weighed average of these three values. The usage of the first value is based on the assumption that the further in the dialog, the smaller the deviation of the average remaining time will be. The second value is used based on the assumption that more data in the model means the averages will be more accurate overall. The third value is again based on the assumption that more data in the current node means that its predictions will be more accurate.

3.5 Hidden Markov Model with Baum-Welch training

To implement a predictor using a Hidden Markov Model the external library [3] was used. In this library a Hidden Markov Model (HMM) can be instantiated and trained with the Baum-Welch algorithm [6].

In this predictor a HMM is instantiated with `questionIds` as states, the hidden states represent the `Status`, the value that is logged indicating the state of the form system. The HMM is defined by a matrix `A`, an array `Pi` and a probability distribution over the hidden states.

The transition matrix `A` represents for each `a_ij` the chance that question `j` follows `i`. These probabilities are calculated by their relative frequencies via n-grams. For example: Let's say the last question asked was a. To compute the transition probability P from a to b one divides the number of times sequence $< a,b >$ has been encountered (bi-gram) by the number of times sequence $< a >$ has been seen (uni-gram). This gives $P(b|a)$. Only uni and bi-grams are used, so the HMM does not use earlier answered questions. These are stored in hash tables for fast lookup. The probability array `Pi` represents the chance that a state is an initial state. The hidden states are `MISSING_DATA`, `OK`, `DATA_ERROR`, `HOLD` and initially with a uniform probability distribution.

The now gotten HMM can be trained using historical logs with the Baum-welch algorithm [6], and is also implemented in [3]

When new traces are available the n-gram tables need not to be updated. The HMM can be tweaked using the Baum-Welch algorithm. When new questions are added in the traces the whole model needs to be retrained, due to the then gotten incorrectness of the states and transition probabilities.

3.6 Aggregator

Ranking the predictions is done by comparing their respective confidence measures. Of course, this is not simply a straight-forward argmax function. Vote weights have to be estimated, so that the confidence measure of each component truly reflects its chance to be correct, in a way that it is comparable to the other components' confidence.

3.7 Use Case Realization

The requirements for the use cases specified in section 1.3 have been satisfied in our interface as such:

1. *Flexibility*
   Since all predictors implement the Predictor interface, the new algorithm can be used anywhere the old ones could, including aggregators.

2. *Continuous training*
   These methods are available:
   `model.train(initialData);`
   `model.train(newData);`
   The Predictor interface specifies that repeated calls to the train() method simply add new data to the model; no data is thrown out and no progress is lost (this potentially limits the choice of algorithms).

3. *Confidence values*
   The predict method in the Predictor interface returns a confidence value along with the prediction itself.

4. *Aggregation*
   It is possible to build aggregators that implement the Predictor interface. This implementation would be no different in use from using other predictors.

5. *Serialization*
   Each Predictor should be forced to implement the `java.io.Serializable` interface and should be responsible for storing its own data that way. There was no time to realize this, but can still be realized without too much trouble.

6. *Concurrency*
   Possible but potentially unsafe. There is no direct support for concurrent execution.

3.8 Evaluation

The predictors were evaluated on data that was collected at Be Informed (the *collected* set). It was collected on

a form that decided if the user was eligible for a visa to the Netherlands. This set contained 508 data points. The data was entered by employees at Be Informed, not by the real target audience of the form. However, the structure and types of questions are similar to those found in real situations, so it is helpful to test the predictors on this dataset.

As the usage data might be too homogeneous, and to be able to study the effects of the order in which data becomes available in the dialog, we additionally conducted some evaluation on designed data with a higher spread when it comes to filling in speed and levels of skipped questions (the *designed* set). This set consists of two types of traces:

1. Depending on the value of one question, all questions take twice as long to answer. The total number of questions is constant across all traces.

2. Depending on the value of one question, a short or long trace is followed. Neither the number of questions nor the total time are constant across traces.

In the second case, the question that determines which way the form forks is not necessarily the question right before the fork (e.g. question 3 determines long or short, but there is always 10 questions asked before the form splits). This was done to measure if the algorithm picks up on the information as it becomes available. This set consists of $10,000$ traces.

Finally, all algorithms were designed to provide a confidence measure along with their results. This is important, because it influences how the application relays its predictions to the users. This confidence measure has been evaluated by determining the probability of a 'correct' answer for each confidence value. It is expected that for predictions where the predictor gives a confidence of 0.4, the probability of getting a correct prediction is roughly 0.4 as well. To determine this probability, a static threshold t was selected. Any prediction with an error higher than t is labeled incorrect, and predictions with a lower error are labeled correct. The value of the threshold t influences the results, and should be chosen carefully based on the application. For the results in this report, it was set at $100s$ (almost two minutes) for time predictions and 1 step for step predictions.

In all cases, the data was split in a train and test set. The generated set was generated only once, and reused for all algorithms to provide a fair comparison.

4 Results

4.1 Collected Set

On the collected set, the predictors outperform the baseline by far on time prediction as well as remaining steps prediction (see figures 11 and 12). Note that the predictors were trained on less than 500 observations. Especially the average tree algorithm really suffers from a lack of data.

4.2 Generated Set

The algorithms also outperformed the baseline implementation on the generated test data (see section 3.8), although the gain is not entirely obvious from the graphs (figure 2, 5, 8). Different parameters were tested for both algorithms (figure 4, 7, 10, 3, 6 and 9).

4.3 Confidence

The algorithms have also been evaluated on the confidence they provide (see figure 13). It follows from section 3.8 that any linear relationship between the probability of correct answers and confidence is desirable.

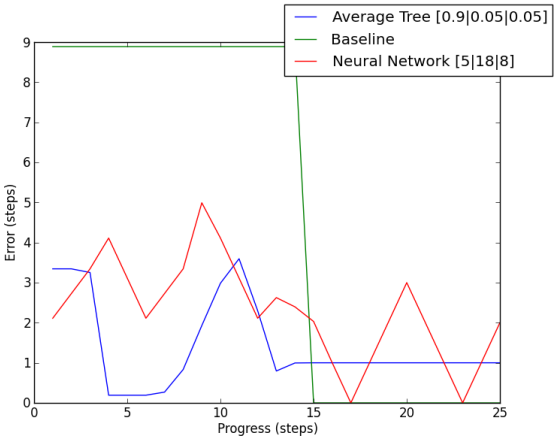

Figure 2: Test data with fork variance (fork at question 10) – Time Prediction

5 Conclusion

The results were a big improvement over baseline results, while providing meaningful confidence values. The average tree algorithm gives the best results on all tests (including confidence values). The other two algorithms (neural networks and hidden Markov models) perform well on the collected set, but not as good on the generated set. Overall the implemented models have the ability to provide Be Informed's users with better predictions for a better user experience.

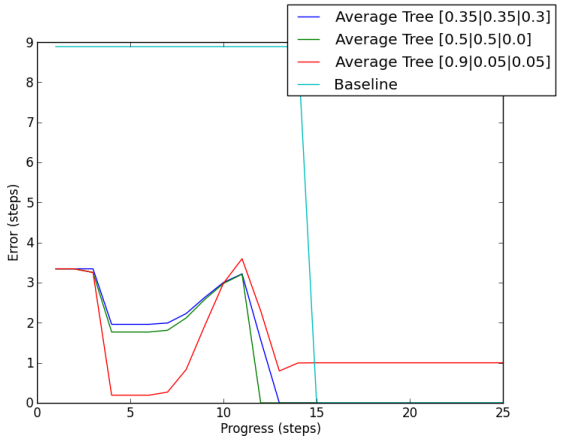

Figure 3: Test data with fork variance (fork at question 10) – Steps Prediction

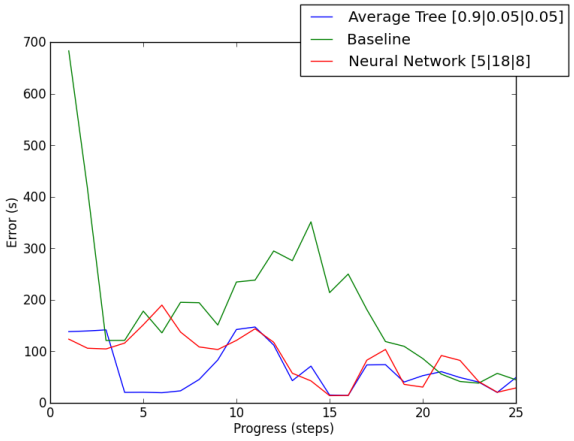

Figure 5: Test data with fork variance (fork at question 10) - Time Prediction

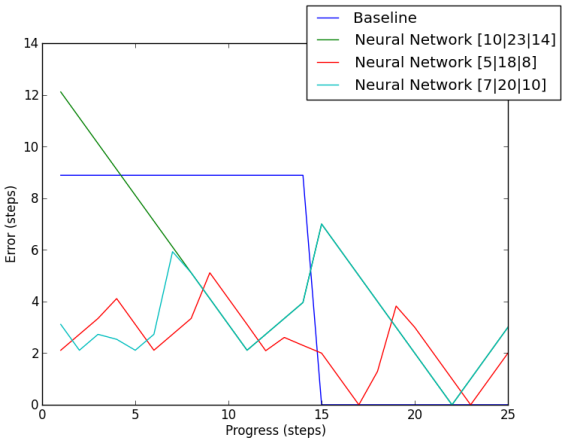

Figure 4: Test data with time variance (fork at question 10) – Steps Prediction

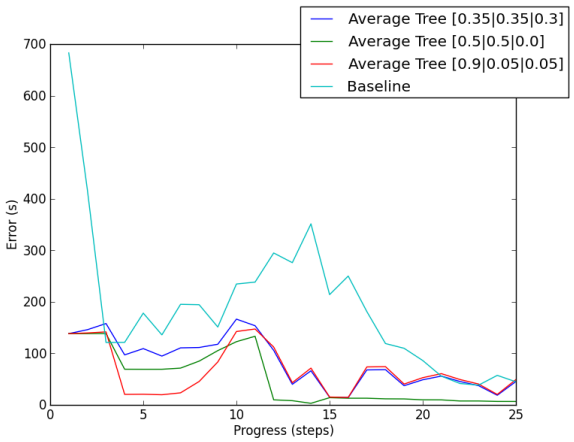

Figure 6: Test data with fork variance (fork at question 10) – Time Prediction

6 Discussion

6.1 Results

In figure 8, the question that determines the duration of the form is the third question in all forms. It is expected to see a drop in error from the learning algorithms at this point. The average tree algorithm does show this drop, but goes back to its old error after a few questions. This is likely due to encountering a question with an answer that was unseen so far. When this occurs, the algorithm is unable to look at answers and reverts to looking at questions only, explaining the drop in performance.

The baseline performance from step 15 onwards in figure 2 and 5 can be explained from the fact that any form with over 15 questions, has the same amount of questions. Since the baseline always guesses the worst-case scenario, it is always right from that point on. A similar fact applies in figure 8. Since the baseline always takes the average time per question so far and multiplies it with the number of questions, it will approach the correct answer very fast for a constant number of questions. The fact that some traces take twice as long is irrelevant, because it only looks at the current trace for its time predictions.

It is also interesting to note that the neural networks lines show roughly the same shape as the average tree lines. This is unexpected, because the algorithms work in a fundamentally different way.

The only point on which the algorithms score poorly is their confidence measure. Although the graph (figure 13) is highly dependent on the threshold value c (see section 3.8), almost none of the confidence graphs are actually linear. Further research should be done to improve these results. For the average tree algorithm, an obvious way

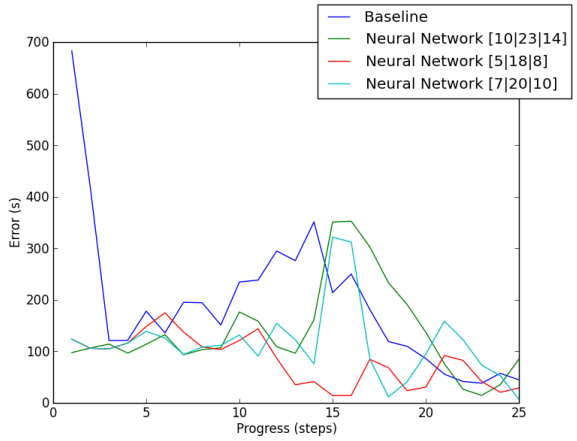

Figure 7: Test data with fork variance (fork at question 10) – Time Prediction

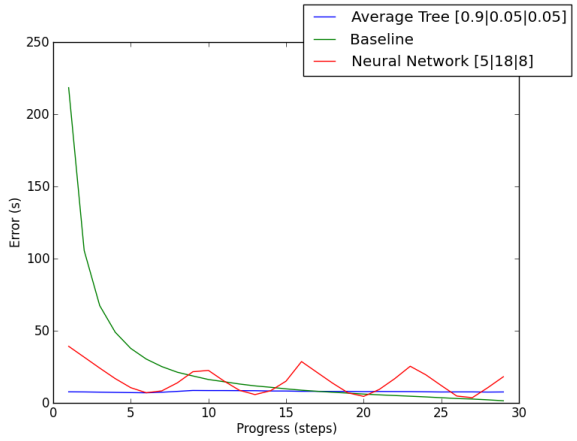

Figure 8: Test data with time variance (fork at question 10) - Time Prediction

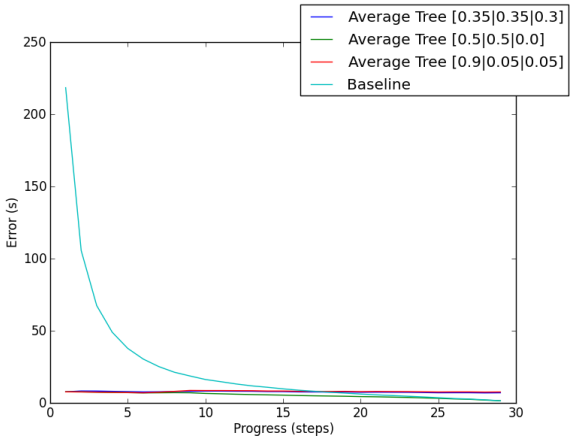

Figure 9: Test data with time variance (fork at question 10) – Time Prediction

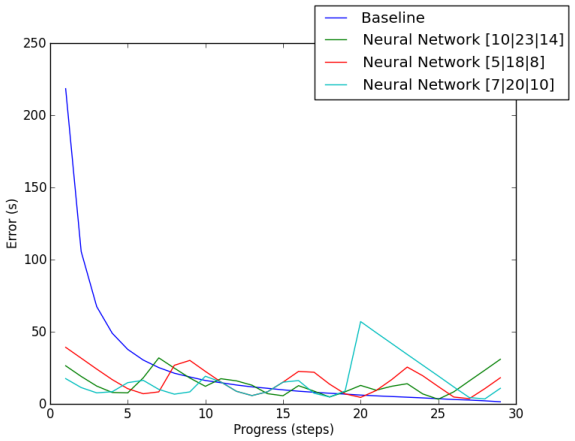

Figure 10: Test data with time variance (fork at question 10) – Time Prediction

to improve the confidence is to look at the 'spread' of the data (i.e. the standard deviation, outliers, etc.) and base the confidence on that. There was not enough time during the project's month to finish these kinds of metrics.

Hidden Markov Model with Baum-Welch

The Hidden Markov Model (HMM) predictor was implemented late in the process and a lot of improvements can still be made. As the results show, it only predicts steps. Although performance is alright, it has its true potential in forms where a lot of different paths can be taken to complete a form.

With initialization of the transition matrix using tri-grams can possibly be achieved much higher performance. This is not implemented at the moment due to use of a library and inability to change the workings of the model. The model would then look at any n states before to get its transition probabilities, instead of always using $n = 2$.

To add time prediction one would calculate the average time taken per question and multiply that with corresponding predicted step, then sum all.

Confidence is also not implemented due to use of an external library of which one cannot extract the probability of the predicted path. When implemented by Be Informed this would not be a problem. One should carefully think about how this probability affects confidence.

6.2 Use Cases Realization

Use cases 1 to 3 are realized.
Use case 4, the aggregation of predictors can be easily implemented and is advised to develop before implement-

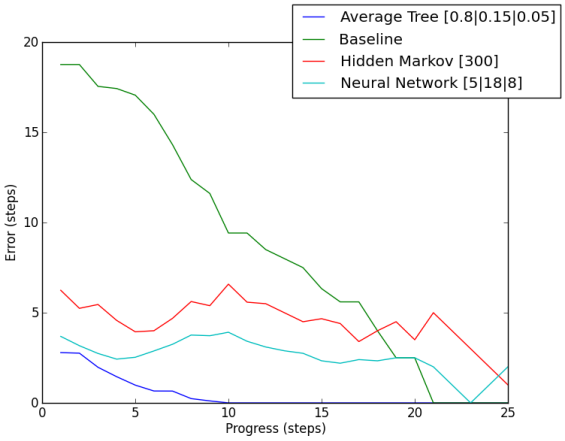

Figure 11: Real Data – All Predictors – Steps Prediction

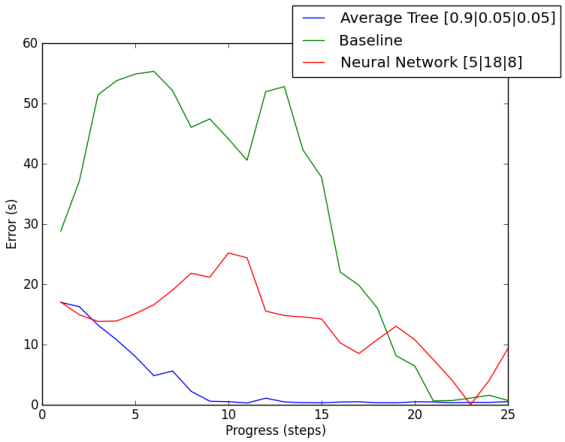

Figure 12: Real Data – All Predictors – Time Prediction

ing more predictors. When real data will be generated a better view on performance of the implementations will be visible, and hence the predictors can be aggregated compared to the baseline.

Use cases 5 and 6 are not realized. This project had the focus on research and not deployment. These use cases have their roots in the latter. Although they are necessary for use in real-world application, they were beyond the scope of this project.

6.3 Further Improvements on Product

The confidence measure is determined by t (see section 3.8), this parameter is not extensively tested on data. To compare techniques better, t could be altered.
The confidence measures given per algorithm are not comparable because they are not calculated the same way. The easiest way to make them comparable is to train confidence on test results. If test results are good confidence

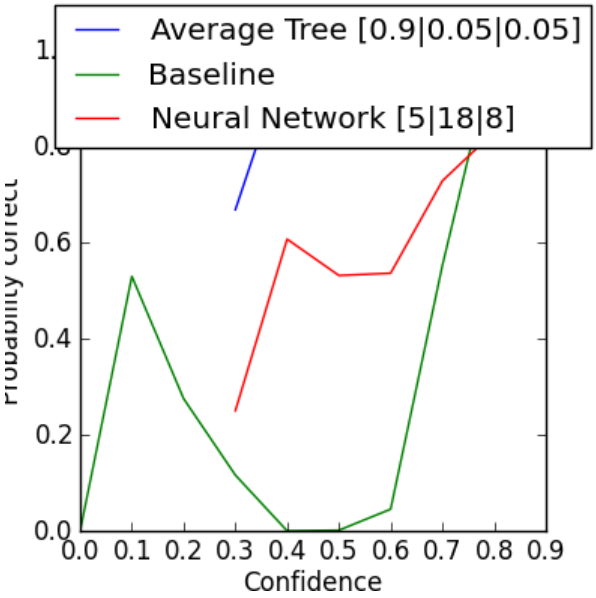

Figure 13: Confidence performance on test data

should be high, if results are bad, confidence should be low. A kind of aggregator could be written for this, so all confidence measures are calculated the same way and thus be comparable.

In each of the algorithms parameters are present. These must be tweaked on a cross validation set for each form they are going to be used on. Also for this a kind of helper class could be written. We did not tweak parameters on cross validation sets because this is something one does when applying a machine learning algorithm on a real specific problem.

6.4 Next Best Algorithms

In section 2.3 the algorithms that have theoretical potential and are not implemented are listed. We advice as next best the Relevance Vector Machine (RVM) and Naive Bayes Classifier (NBC), not necessarily in that order.

The RVM is proven useful in the estimation of remaining useful life of batteries [8], and because this is a similar problem this is a potential candidate. The drawback of local maxima can be overcome by extensive testing and use of a cross validation set to find the optimal parameters. We did not implement it because the preprocessing is complicated.

The NBC is interesting because it trains fast and is easy to implement. Next to that, the NBC resembles neural networks and this has been proven to be well performing.

The Hidden Markov Model still need to be tested, but if it performs well, Viterbi could be implemented. This

consumes a lot less space and time and possibly could perform as good.

The Random forest is not recommended. The Average Tree shows good performance and in our vision the Random Forest is just more work for the same result.

Acknowledgments

Thanks to:

- Jeroen van Grondelle, research director at Be Informed
- Bert Bredeweg, assoc. prof. at the University of Amsterdam
- dr. Maarten van Someren at the University of Amsterdam

References

[1] Breiman, L. Random Forests. *Machine Learning*, 45:5–32, 2001. doi: 10.1023/A:1010933404324.

[2] Hastie, T. & Pregibon, D. Shrinking Trees. 1990.

[3] Jean-Marc Franois. Jahmm - Hidden Markov Model (HMM). An implementation in Java [software package]. `http://www.run.montefiore.ulg.ac.be/~francois/software/jahmm/`.

[4] Jeff Heaton e.a. Encog Machine Learning Framework [software package]. `http://www.heatonresearch.com/`.

[5] Jeroen van Grondelle & Geert rensen. Towards Webscale Business Processes, 2013.

[6] L. E. Baum. An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.

[7] Pandey, S. & Nepal, S. & Chen, S. A Test-bed for the Evaluation of Business Process Prediction Techniques. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 7th International Conference*, 2011. doi: 10.4108/icst.collaboratecom.2011.247129.

[8] Saha, B. & Goebel, K. & Christophersen, J. Comparison of prognostic algorithms for estimating remaining useful life of batteries. *Transactions of the Institute of Measurement and Control*, 31: 293–308, 2009.

[9] Tipping, M. E. Sparse Bayesian Learning and the Relevance Vector Machine. *Journal of Machine Learning Research*, 1:211–244, 2001. doi: 10.1162/15324430152748236.

[10] Robert L. Welch. Real time estimation of bayesian networks. *CoRR*, abs/1302.3609, 2013.

[11] Werbos, P. J. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[12] Yeon, J. & Elefteriadou, L. & Lawphongpanich S. Travel time estimation on a freeway using Discrete Time Markov Chains. *Transportation Research Part B*, 42:325–338, 2008. doi: 10.1016/j.trb.2007.08.005.

A Long list

A.1 Neural Network with Back Propagation

Neural networks are derived from the workings of the human brain. They are strong, but in a computer slow. It is hard to exactly understand how it is they work, but often they do pretty well. A neural net has one input layer of neurons which are not connected. A hidden part, which can have multiple connected layers, which consists of nodes that have connection input from each previous layer node, and have output connections to each next layer node. The output layer consists of unconnected output neurons. The nodes have activation functions, thresholds, if activation reaches this point, it fires output. One can choose the function to use in the network

Input

Back propagation takes a normalized numerical vector as input. For prediction of progress in a dialog system, the input vector consists of the answers the user has provided for each of the questions (method 1). Each dimension of the input vector corresponds to a specific question. The order in which the questions were answered is lost in this process.

Open questions are disregarded in the normalization process (it is a different and difficult problem to make natural language useful). Open ranges (such as age or salary) can easily be normalized by dividing every value by the maximum (as found in the data). Questions with a fixed set of possible answers are given a numeric value, and then normalized as normal ranges. All values will be offset by 1. Questions that have not yet been asked will get the value 0

Training

The network is trained with back propagation (see [11]). Back propagation is supervised. Examples are given with desired output vector where each dimension corresponds to a bucket (i.e. 1-3 minutes, 3-5 minutes, etc.). The size of these buckets is a parameter to the algorithm. While training, one of the dimensions of the output vector is 1, the others all 0.

Output

The algorithm produces a vector with the same structure as what it was trained on, but each dimension will contain a value between 0 and 1. This value can be interpreted as the probability that the input trace belongs to that bucket. The output will be the range associated with the bucket with the highest probability, and the confidence will be the exact value of that dimension in the output vector.

Advantages

- Low (almost no) storage needs; the algorithm only updates its matrix every time it trains.
- Gives a degree of certainty (i.e. a probability per bucket)
- the chances from the output vector could be used as a confidence measure
- easy to update when new data is available
- prediction is usually very fast

Disadvantages

- Neural networks are hard to understand and comprehend.
- Backpropagation training is slow

A.2 Average Tree

The average tree is a kind of decision tree. These have a all tree restrictions that hold for trees in AI.

Input

The average tree algorithm uses questions and appurtenant answers as input.

Training

Two trees are built. One consisting of nodes representing questions. Only question sequences are represented in this tree. The other tree is built by laying out all question-answer pairs as nodes, where different answers branch to different subtrees.

Output

The algorithm outputs a weighted average of three predictions:

- The average remaining time in the path including answers
- The average remaining time in the path excluding answers
- The average remaining time at the parent node in the path excluding answers (minus the amount of time the last question took)

These weights are partly input to the algorithm, but they are also adjusted based on the amount of data in each of these nodes (weighing more common paths heavier).

The confidence output is based on the assumption that the model gets better with more data, but also that the averages should get more accurate the longer the path gets (since there is less interference from other branches).

Advantages

- Simple implementation and training
- Fast prediction and training
- Gets better with more data
- Provides confidence values

Disadvantages

- Requires a lot of data to handle uncommon paths
- Might be very inaccurate if there is a lot of variation in the branches

A.3 Markov Model with Viterbi

A Markov Model assumes that in a path only the last n states in the path are relevant for the next, where n is a constant, usually between 1 and 5.

Input

The Markov Model uses questions and appurtenant timestamps as input.

Training

The average time of each encountered path, or question traces, in the historical logs is calculated. Then transition probabilities, represented in the transition matrix A, represents for each $a_i j$ the chance that question j follows i, is calculated by relative frequencies. This could be done by looking at n-grams.

Output

With Viterbi's algorithm (see Yeon, J. & Elefteriadou, L. & Lawphongpanich S. [12]) the most probable path is calculated given a last answered question, a state. This path consists of remaining questions which have, based on the historical logs, an average time, which, in their turn, must be added to result in a time prediction. The confidence on this prediction would be generated by comparing the total path probability to path length and other paths' probabilities.

Advantages

- Reliable, has been proven to be an effective path finding method in many fields
- Easy to train on new data

Disadvantages

- Will need a lot of data for a reasonable result
- Will probably need some smoothing in order to account for unseen paths

A.4 Hidden Markov Model with Baum-Welch training

A Hidden Markov Model is a Markov Model with hidden states. In this case each state has next to a transition probability matrix for next states, a emission probability that defines the probability that a hidden state occurs corresponding to concerning state.

Input

The questions and their return values are input. The states are all possible questions, the hidden states represent the return value.

The transition probabilities are calculated in the same manner as above with the Markov Model. The emission probabilities are initialized using uniform distribution. The observation emission matrix B represents the probability of observing O_t at a certain state (see for a full description [7]).

Training

The Baum-Welch algorithm is used to train the model using return values of all possible questions, in the same order as the states are in. This means some preprocessing of the historical logs since the questions can occur in varying orders.

Output

A HMM, described by matrices A and B is output from training. Confidence and a prediction are obtained the same way as with Viterbi, but now with the Forward algorithm.

Advantages

- A tested method for prediction problems
- Fast training

Disadvantages

- Cannot be tweaked with new data

A.5 Naive Bayes Classifier

The Naive Bayes classifier is the only unsupervised learning algorithm on the list. It is called naive because it assumes unconditionally of probabilities. (see [10])

Input

Features are individual questions (as with random forest). The user's answers are values for these features. The classifier does not care in what form these answers are (it simply looks at unique values).

Testing

The classifier is tested on buckets of time (i.e. 3-5min, 5-7min etc.). The algorithm finds the probability of each feature's value given each class.

Output

The algorithm outputs a probability of the trace belonging to each class (i.e. $p(t|C)$), where classes are the buckets. The bucket range of the class with the highest probability is returned. The confidence is the probability the algorithm output for this class.

Advantages

- Training is fast
- Estimation is fast

Disadvantages

- No built-in confidence measure

A.6 Relevance Vector Machine

The Relevance Vector Machine (RVM) is an extension of a support vector machine (SVM). An SVM in its traditional form is a binary classifier or gives a point as output, as would regression. The extension to RVM consists of abolishing some constraints the SVM has.

Input

Input are n data points x_n, t_n where x_i is a vector, like the one described in he section back propagation, of length m ($m = maxnumberofquestions$) and t_i the target, the buckets of time ranges or remaining question ranges. To add probability noise is added by a Gaussian over t_n with mean $y(x_n)$ and variance σ^2, thus $p(t_n|\vec{x}) = \mathcal{N}((t_n|y(x_n), \sigma^2)$.

Training

Training takes place as with a regular SVM. See Tipping Tipping, M. E. [9] section 2.3.

Output

The output of the RVM consists of normal distributions that describe the probability of a partial question trace belonging to a bucket. This probability can be interpreted as a confidence measure, the flatter the less confident.

Advantages

- Outputs probabilities that can be interpreted as confidence measures
- Can be tweaked with new examples

Disadvantages

- Slow training

A.7 Random Forest

A variant on decision trees that uses multiple trees to find the best classification (see [1] for a complete description).

Input

Input features are the questions (such as 'what is your age'). Their values are as they appear in the data. Data does not need to be normalized for use in a decision tree. It is easy to add different features (such as total time until now), as there is no structural difference with questions/answers.

Training

The tree is trained on buckets of times (i.e. 3-5 minutes, 5-7 minutes, etc.). Only complete traces are used during the training process. The algorithm also stores probabilities of each branch at every node (e.g. 70% for age < 65, 30% for age $>= 65$). These probabilities can be based on simple relative frequency measures, but also more complex Markov models (i.e. based on the last three asked answers). They are used for prediction of incomplete traces. The trees generated by the random forest do not need to be pruned, as similar behavior emerges from the voting of trees by the algorithm.

Output

The tree outputs a bucket of time. If the tree is walked without a value for each feature (not all questions have always been answered), at each node where no decision can be made, the one with the highest probability will be used.

Advantages

- Can be modified to provide confidence measures

- At least as fast as normal decision trees (but more accurate)
- Can handle two-valued input

Disadvantages

- Might not be able to continuously train

B Architecture