CONTENDER

D. Suendermann, J. Liscombe, R. Pieraccini

SpeechCycle Labs, New York, USA
{david,jackson,roberto}@speechcycle.com

ABSTRACT

Contender (or what the academic community would refer to as a light version of *reinforcement learning*) is a simple technique to experiment with a number of competing paths in a (commercial) spoken dialog system. By randomly routing certain portions of traffic to individual paths and computing average rewards for each of the routes, the goal is to find out which one performs best. This paper is to do away with common uncertainties on how to set up contender weights, how much data needs to be accumulated to draw reliable conclusions, and how this all relates to the notion of *statistical significance*.

Index Terms— Contender, commercial spoken dialog systems, statistical significance

1. INTRODUCTION

Most often, commercial dialog systems have to follow severe contractual restrictions as to how they exactly need to behave. Business stakeholders and application architects decide upon how prompts are phrased, when callers are offered to speak to a human agent, what items are presented in menus, or that certain caller requests need to be possible at all time during the dialog (such as "agent", "help", or "go back") independently of whether callers actually ever say such things. In a bureaucratic manner, requirements, design, as well as voice user interface specifications are defined and are then rigorously implemented.

Considering that high-trafficked commercial applications are built primarily to

(A) save human operator handling time while

(B) delivering a good user experience,

most of the design principles used to build these applications focus on positively impacting (A) or (B).

An example: We are looking at a cable TV troubleshooting application replacing human agents in certain situations [1]. The original application performed an automated reboot of a dysfunctional cable box first, and, in case this did not fix the problem, it instructed the caller how to do a manual reboot. At some point, the manager of the call center deploying this application suggested to reverse the reboot order, i.e., first manual, then automatic, since he said his center's human operators were doing so, and he was convinced this to be the optimal order.

Traditionally, and depending on the position of the requesting party, this kind of requests are implemented without further questioning. In a post-mortem analysis, the performance impact of the change may be measured to prove (or possibly disprove) the initiator's arguments. However, in doing so, it happens often enough that a number of disparate performance metrics are consulted until one proving the initial argument is found. E.g., in the above example, the original implementation may have resulted in a higher automation rate than the new one, whereas the latter produced lower average handling times. Automation rate and average handling time are typical opositional metrics, since failed automation often results in short calls, as opposed to automated calls often taking a considerably longer duration[1].

Even if the performance metric is not up for discussion, a comparison of average performance before and after the fact may not be very reliable. This is simply because of the time dependence of performance due to seasonal deviations and special events such as outages, promotions, technical changes, etc. For example, a recent change to the backend services of a certain application showed a performance gain of 1.5 percentage points, but the responsible project manager asked to be careful with this result since he had reported similar improvements during the summer season of previous years without any changes to the application at those times.

A viable solution to this problem seems to be the simultaneous deployment of the two systems in comparison and the routing of reasonable portions of traffic to both of them. Similarly, a single system can be deployed containing a splitting module that, for every call, randomly selects one of its outbound transitions (paths). Each of the paths leads to an implementation of one of the competing strategies. This approach, called *contender* [2], overcomes the time dependence but comes along with some other questions. In the world of quick business decisions, already hours after a contender went into production, decision makers would like to know which of the systems is the winner. Naturally, responsible analysts will

[1] In our example application, automated calls took 210 seconds on average whereas non-automated calls took 139 seconds measured on 7,207,925 production calls.

push back saying that, after a couple of hours, results are not yet reliable, and one has to wait for a certain degree of *statistical significance*. So, when are results statistically significant? We have heard people responding things like

- After two weeks.
- After 10000 calls.
- When the performance difference is at least 1%.

Even if one of the above answers is correct (which they certainly are not for many possible scenarios), there is some more questions not yet answered such as

- How do these answers vary for contenders with more than two paths?
- Is it possible to vary the amount of traffic going down the path prior to achieving the required statistical significance in an attempt to optimize the gross average performance of the deployment including the contender stage?

This paper is to answer all the above questions in a mathematically sound fashion. In Section 2, we will briefly reason about the reward function being the fundamental requirement for a contender analysis to take place. We will then investigate what exactly statistical significance means in a contender scenario in Section 3. Finally, in Section 4, we will prove the impact of a dynamic adjustment of the traffic going down contender paths. In Section 5, we will summarize our findings and list points we have not yet been able to completely resolve.

2. REWARD FUNCTION

To avoid situations like the one in our introducing example, where constant arguments about the performance metric to be used prevent a contender experiment from being conclusive, all involved parties have to agree on a single scalar metric essential for all the further considerations of this paper. A typical scalar performance metric is the fact whether a call was automated ($A = 1$) or not ($A = 0$). This type of scalar performance metric is also referred to as *reward* as done in many reference publications on machine learning of dialog management strategies [3, 4] as well as reinforcement learning [5][2]. Hence, if we would want to express a call's performance solely based on whether it was automated or not, our reward function was

$$R = A \; . \tag{1}$$

Certainly, a single raw metric such as whether a call was automated, the handling time, the number of operator requests, whether the caller hung up, or the number of recognition failures, to name only a few, is not exactly what the the application analysis team was searching for. Rather, the actual reward function can be any type of combination between these metrics, e.g. a linear combination. If, for instance, only financial arguments are to be considered (principle (A) in the introduction), a derivation of a call's reward could read as follows:

- Every automated call prevents a human operator from handling the call and consequently saves a certain average amount R_A.
- On the other hand, every minute of running a spoken dialog system generates the cost R_T for hosting and license fees, telephony, energy, hardware, maintenance, and so on.

For this simple derivation, the savings (or reward) R produced by a call with the duration T are

$$R = R_A A - R_T T \; . \tag{2}$$

As we will see in Section 3, to be able to estimate the statistical significance of results, we will use a parameterized model of the probability density function of the reward, $f(r)$. Let us first derive f for the simple case of the automation-dependent reward (see Equation 1). When we consider there is an (unknown) probability $p_A := p(A = 1)$ that a call will be automated, then we know that $R = 1$ with probability p_A and $R = 0$ with probability $1 - p_A$. Consequently, we obtain the density function

$$f(r; p_A) = (1 - p_a)\delta(r) + p_a\delta(r - 1) \; . \tag{3}$$

Now, we want to model f for the case of the financial reward function (Equation 2). This is a little more tricky since it requires us to model the probability density of the call duration for both automated and non-automated calls. As a typical probability model for handling times, we want to use the Gamma distribution here. It is defined as [6]

$$g(t; \alpha, \beta) = \begin{cases} \frac{\beta^\alpha}{\Gamma(\alpha)} t^{\alpha-1} e^{-\beta t} & : \quad t > 0 \\ 0 & : \quad \text{otherwise} \end{cases} \tag{4}$$

with the two parameters α (shape parameter) and β (rate parameter)[3]. Figure 1 shows a duration distribution of non-automated calls to the example cable TV troubleshooting system measured on the most recent 622,900 calls handled (July 2010). It also shows the best-fitting Gamma distribution according to the least squares method.

In order to derive the density function for the reward function as a whole, we have to consider

- the probability of automation p_A, and

[2] In these disciplines, rewards are usually accumulated over the multiple interaction turns of a call, whereas in the scope of our present work, we are only interested in the final reward of the call.

[3] Since t is a time variable, β has the unit s^{-1}, so has g.

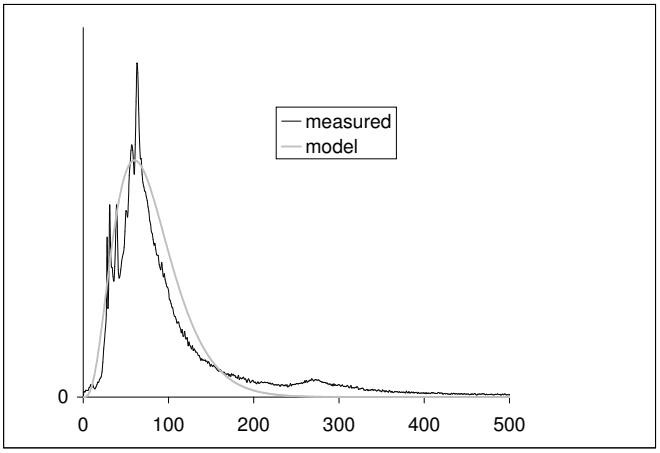

Fig. 1. Fitting a Gamma distribution to match the probability density of call durations.

- the fact that the distribution paramaters can strongly vary between automated and non-automated calls; so we will distinguish between α_A, β_A and α_N, β_N.

This leaves us with five free parameters altogether and the combined *bivariate* density function

$$f(r; p_A, \alpha_A, \beta_A, \alpha_N, \beta_N) = \tag{5}$$

$$p_A \cdot g\left(\frac{R_A - r}{R_T}; \alpha_A, \beta_A\right) + (1 - p_A) g\left(-\frac{r}{R_T}; \alpha_N, \beta_N\right) .$$

3. STATISTICAL SIGNIFICANCE

3.1. Why does contender not use common statistical hypothesis tests?

Once a contenderized application goes into production, traffic is routed down all contender paths. Each of the processed calls is associated with one of the I paths as well as certain reward observations:

$$\mathbf{R}^i = \{r_1^i, r_2^i, \ldots\} \text{ for } i \in \{1, \ldots, I\} . \tag{6}$$

Now, rather than computing the average performance for all elements of $\mathbf{R}^i$ to determine the winning path, we want to follow an approach whereby we estimate how likely it is that path i is the actual best-performing path. This approach follows the principles of statistical hypothesis testing [7] in that it aims at estimating probabilities for certain hypotheses. However, after exploring the properties of common test statistics such as t- or z-test, it turns out there are several reasons we cannot apply them to our current work:

- These statistics can only be applied when comparing exactly two competitors.
- Statistical tests involving more than two competitors such as ANOVA [8] or Tukey's test [9] compare all competitors in a pair-wise fashion to find out which ones significantly differ from one another. This is again not applicable to our scenario which is to determine a single probability for a given competitor to be the winner.
- The test statistics are imprecise due to certain assumptions such as that
  - the reward follows a univariate normal distribution (this is not the case as we saw in the examples of Section 2),
  - that there is a minimum number of samples per path,
  - that the variations of the compared univariate normal distributions are either known or equal each other.

3.2. Estimating winning probabilities

Taking all common test statistics aside, let us now attempt to estimate the winning probability of a contender path using raw mathematical means and our knowledge about the reward density.

To begin, we look at a single contender path and assume we knew its optimal parameterization $\mathbf{a}$ which is the set of all parameters of the reward function model. The respective probability density for a single observed call is $f(r; \mathbf{a})$. If we had processed exactly two calls resulting in the reward set $\mathbf{R} = \{r_1, r_2\}$, there were two orders in which these events could have happened, and the probability density combination would have been

$$f(r_1, r_2; \mathbf{a}) = 2 f(r_1; \mathbf{a}) f(r_2; \mathbf{a}) . \tag{7}$$

Generally, we can write

$$f(\mathbf{R}; \mathbf{a}) = c \prod_{r \in \mathbf{R}} f(r; \mathbf{a}) \tag{8}$$

with the normalization constant c. Now, it is time to get a second contender path into play. Using the above derivations, we want to know how likely it is that the expected reward of path 1 given its observation set $\mathbf{R}^1$ is greater than that of path 2 given $\mathbf{R}^2$. In doing so, we have to consider (i.e. integrate over) all possible model parameterizations for both paths all of which could potentially have produced $\mathbf{R}^1$ and $\mathbf{R}^2$:

$$p(r_1 > r_2; \mathbf{R}^1, \mathbf{R}^2) = \tag{9}$$

$$\int_{\mathbf{a}^1} f(\mathbf{R}^1; \mathbf{a}^1) \int_{\mathbf{a}^2} f(\mathbf{R}^2; \mathbf{a}^2)\, \varepsilon(\mathbf{a}^1, \mathbf{a}^2)\, \mathrm{d}\mathbf{a}^1 \mathrm{d}\mathbf{a}^2$$

$$\text{with } \varepsilon(\mathbf{a}^1, \mathbf{a}^2) = \begin{cases} 1 & : \quad E(r; \mathbf{a}^1) > E(r; \mathbf{a}^2) \\ 0 & : \quad \text{otherwise} \end{cases} \tag{10}$$

where $E(r;\mathbf{a})$ refers to the expected value of r given the model **a**. To know $p(r_1 > r_2)$ in a two-path contender, directly leads to the winning probabilities

$$p(1) = p(r_1 > r_2) \text{ and } p(2) = 1 - p(r_1 > r_2)\,. \quad (11)$$

In order to extend this derivation to multiple paths, we make use of the fact that a contender winner i is supposed to outperform all other competitors $j; j \neq i$. In the list of all $I!$ possible performance rankings, there are $(I-1)!$ rankings with i at the top. E.g., given a four-path contender, we have $(4-1)! = 6$ scenarios for path 3 to be the winner:

$$\begin{array}{cccc} 3 & 1 & 2 & 4 \\ 3 & 1 & 4 & 2 \\ 3 & 2 & 1 & 4 \\ 3 & 2 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 3 & 4 & 2 & 1 \end{array}$$

The probabilities for all these scenarios have to be summed up to yield ith winning probability

$$\begin{aligned} p(i) &= p(r_i > r_1 > r_2 > \cdots > r_{I-1} > r_I) \quad (12) \\ &+ p(r_i > r_1 > r_2 > \cdots > r_I > r_{I-1}) \\ &\vdots \\ &+ p(r_i > r_I > r_{I-1} > \cdots > r_2 > r_1) \end{aligned}$$

where the individual addends can be calculated via an extension of Equation 9:

$$p(r_i > r_j > \cdots > r_k; \mathbf{R}^1, \ldots, \mathbf{R}^I) = \quad (13)$$

$$\int_{\mathbf{a}^i} f(\mathbf{R}^i; \mathbf{a}^i) \int_{\mathbf{a}^j} f(\mathbf{R}^j; \mathbf{a}^j) \cdots$$

$$\int_{\mathbf{a}^k} f(\mathbf{R}^k; \mathbf{a}^k)\, \varepsilon(\mathbf{a}^i, \mathbf{a}^j, \ldots, \mathbf{a}^k)\, \mathrm{d}\mathbf{a}^i \mathrm{d}\mathbf{a}^j \ldots \mathrm{d}\mathbf{a}^k;$$

$$\varepsilon(\mathbf{a}^i, \mathbf{a}^j, \ldots, \mathbf{a}^k) = \begin{cases} 1: E(r;\mathbf{a}^i) > E(r;\mathbf{a}^j) > \cdots > E(r;\mathbf{a}^k) \\ 0: \qquad\qquad \text{otherwise} \end{cases}$$

3.3. An example

While the above formula apparatus may seem overly complex and probably gives a slight flavor of intractability, computational complexity is generally not a problem here. This is mainly because of the very limited number of contender paths in most commercial deployments rarely exceeding five or so. Furthermore, calculation complexity can be largely reduced by using simple reward functions such as the automation-based one (see Equation 1). To prove this point, let us consider a 3-path contender and quickly re-iterate over all the steps explained in Section 3.2.

To get started, it should be pointed out that we have a single parameter $\mathbf{a} = p_A$ per path in the automation reward scenario and that the reward set is composed of only two possible values, 0 and 1. Therefore, we can represent a reward set **R** by the counts of ones (c_1 and zeros c_0). Using Equation 3 for the specific density function, and understanding that the expected reward is equivalent to the actual probability of being automated, $E(r;p_A) = p_A$, allows us to rewrite Equation 8 as

$$\begin{aligned} f(\mathbf{R};p_A) &= c \prod_{r\in\mathbf{R}} f(r;p_A) \quad (14) \\ &= c \cdot (f(1;p_A))^{c_1} \cdot (f(0;p_A))^{c_0} \\ &= \binom{c_1+c_0}{c_1} (p_A)^{c_1} (1-p_A)^{c_0}\,. \end{aligned}$$

The next step is to insert the result into Equation 13

$$p(r_i > r_j > r_k; c_1^i, c_0^i, c_1^j, c_0^j, c_1^k, c_0^k) = \quad (15)$$

$$\binom{c_1^i + c_0^i}{c_1^i} \binom{c_1^j + c_0^j}{c_1^j} \binom{c_1^k + c_0^k}{c_1^k}\,.$$

$$\int_0^1 x^{c_1^i}(1-x)^{c_0^i} \int_0^x y^{c_1^j}(1-y)^{c_0^j} \int_0^y z^{c_1^k}(1-z)^{c_0^k} \mathrm{d}z\, \mathrm{d}y\, \mathrm{d}x\,.$$

Finally, we get the three winning probabilities using Equation 12 (and discarding the annoying parameter arguments):

$$\begin{aligned} p(1) &= p(r1 > r2 > r3) + p(r1 > r3 > r2)\,, \\ p(2) &= p(r2 > r1 > r3) + p(r2 > r3 > r1)\,, \\ p(3) &= p(r3 > r1 > r2) + p(r3 > r2 > r1)\,. \quad (16) \end{aligned}$$

4. ADJUSTING THE TRAFFIC

As we have now learned how to determine the winning probability of a contender path, the question arises on how these findings are to be translated into actions and when to do so. Does a winning probability of 1% mean we should remove the path from the contender? Or 5%, or 10%? What if there were 10 contender paths each of which has a 10% winning probability? What should we do when the progress is extremely slow? 60% probability after one week, 70% after three weeks, 80% after two months?

In Section 3.1, we have introduced our analysis method as a statistical hypothesis test similar to established ones such as t- or z-tests. Those use a p-value and an associated significance level to determine whether a difference is assumed to be statistically different. Common significance levels are 5%, 1%, and 0.1%, so, we could assume similar levels in our present test. However, the argument made above that, the more paths are involved, the lesser the individual probabilities become, suggests that the significance level should be sufficiently small for the task in question.

On the other hand, the more paths are involved, the less traffic is routed to each of them, and the smaller the significance level, the more traffic is required to make a decision. This conflicting finding is yet another argument to investigate whether actions can be carried out even before statistical significance was found. Specifically, it is of interest to see whether the amount of traffic hitting each contender path may be adjusted to positively impact the gross average reward of an application. A possible approach would be to adjust path traffic in a continuous fashion using the winning probabilities themselves as weights. That is, when a path shows a 90% probability to be the winner, we route 90% of traffic down this path.

In the following, we want to find out whether this approach is actually helpful in terms of the gross average reward. To that end, we compare three different approaches:

(1) Randomly chose a winner.

(2) Chose a winner once its winning probability is above a certain threshold.

(3) Dynamically adjust traffic based on winning probabilities.

Before we get started, let us agree on some standards:

- If we would know the contender winner from the very beginning (for example from an octopus), we could achieve the optimal per-call reward $\hat{r}$, the actual infinite-horizon reward of the best performing path.
- Without loss of generality, let us focus on a two-path contender.
- Let Δ be the (positive) performance difference between the actual infinite-horizon rewards of the contender paths.
- The contender deployment starts at time t_0.

Let us get started with (1). If we randomly choose path 1 to be the winner it will be optimal ($r = \hat{r}$) with probability p and sub-optimal ($r = \hat{r} - \Delta$) with $1 - p$. So, the expected reward at time t_0 is

$$\begin{aligned} E_1(r, t_0) &= \hat{r}\, p + (\hat{r} - \Delta)(1 - p) \\ &= \hat{r} - \Delta + \Delta\, p\,. \end{aligned} \tag{17}$$

Analogously, routing all traffic down path 2 would result in

$$\begin{aligned} E_2(r, t_0) &= \hat{r}(1 - p) + (\hat{r} - \Delta)p \\ &= \hat{r} - \Delta\, p\,. \end{aligned} \tag{18}$$

When the decision to route down a certain path is made completely randomly, then we have $p = 0.5$ and, consequently,

$$E_1(r, t_0) = E_2(r, t_0) = \hat{r} - 0.5\Delta\,. \tag{19}$$

Next, we want to explore option (2). Instead of routing random traffic to both paths for the entire time of the deployment, we only do so until we achieve a probability update at time t_1 of

$$p(t_1) = p(t_0) + \Delta_p. \tag{20}$$

If $\Delta_p > 0$, we can now make a hard decision and route all traffic to path 1:

$$\begin{aligned} E_1(r, t_1) &= \hat{r} - \Delta + \Delta(p + \Delta_p) \\ &= \hat{r} - 0.5\Delta + \Delta_p\Delta \\ &> E_1(r, t_0)\,. \end{aligned} \tag{21}$$

Otherwise, we would route the traffic to path 2:

$$\begin{aligned} E_2(r, t_1) &= \hat{r} - \Delta(p + \Delta_p) \\ &= \hat{r} - 0.5\Delta - \Delta_p\Delta \\ &> E_2(r, t_0)\,. \end{aligned} \tag{22}$$

Thus, routing the full traffic to the most probably winning path after waiting until t_1 outperforms the purely random choice (1) as we clearly expected.

Finally, let us look at (3), that is, whether probability-based traffic weighting can be of benefit. Revisiting (2) for a moment, a typical approach would be to leave 50% traffic on both paths until, at a time t_1, a certain statistical significance was found (e.g., $p < 5\%$ or $p > 95\%$) and than to route full traffic to the probable winner. Let us now assume, we analyze the winning probability at a time t' with $t_0 < t' < t_1$ and find, similar to Equation 20,

$$p(t') = p(t_0) + \Delta_p. \tag{23}$$

As opposed to the above example, this time, we make a *soft* decision by routing $p(t')$ traffic to the respective path. If $\Delta_p > 0$, the updated expected reward becomes:

$$\begin{aligned} E_1(r, t') &= (0.5 + \Delta_p)(\hat{r} - 0.5\Delta + \Delta_p\Delta) \\ &+ (0.5 - \Delta_p)(\hat{r} - 0.5\Delta - \Delta_p\Delta) \\ &= \hat{r} - 0.5\Delta + 2\Delta_p^2\Delta \\ &> E_1(r, t_0)\,. \end{aligned} \tag{24}$$

The equivalent can be shown for $\Delta_p < 0$. This proves that an update according to the winning probability is indeed beneficial for the gross reward of an application.

5. SUMMARY AND OPEN QUESTIONS

This paper focused on investigations towards the statistical significance of contender analyses. We have shown a way to estimate precise winning probabilities of contender paths that can be used to

- draw reliable conclusions on which path is performing best, to which degree, and how significant these findings are, and

- to route traffic depending on these probabilities, an approach we showed to outperform the conventional way of making hard decisions once statistical significance is assumed.

There is a number of questions still open with respect to the current work. Among others, we have to investigate

- how contender analysis relates to the field of reinforcement learning,
- how the presented statistical test can be scaled to a large number of contender paths (tractability?),
- how the present analysis can cope with parameter-dependent or interacting contenders,
- how we can (dis)prove that probability-dependent routing is optimal (we have only shown that it outperforms the baseline, but are there even better techniques?),
- and, last but not least, how to overcome some of the paper's mathematical glitches.

6. REFERENCES

[1] K. Acomb, J. Bloom, K. Dayanidhi, P. Hunter, P. Krogh, E. Levin, and R. Pieraccini, "Technical Support Dialog Systems: Issues, Problems, and Solutions," in *Proc. of the HLT-NAACL*, Rochester, USA, 2007.

[2] D. Suendermann, J. Liscombe, and R. Pieraccini, "Minimally Invasive Surgery for Spoken Dialog Systems," in *Proc. of the Interspeech*, Makuhari, Japan, 2010.

[3] E. Levin, R. Pieraccini, and W. Eckert, "A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies," *IEEE Trans. on Speech and Audio Processing*, vol. 8, no. 1, 2000.

[4] S. Young, "Using POMDPs for Dialog Management," in *Proc. of the SLT*, Palm Beach, Aruba, 2006.

[5] L. Kaelbling, M. Littman, and A. Moore, "Reinforcement Learning: A Survey," *Journal of Articial Intelligence Research*, vol. 4, 1996.

[6] R. Hogg, J. McKean, and A. Craig, *Introduction to Mathematical Statistics*, Prentice Hall, Englewood Cliffs, USA, 1995.

[7] M. Schervish, *Theory of Statistics*, Springer, New York, USA, 1995.

[8] K. Mardia, J. Kent, and J. Bibby, *Multivariate Analysis*, London Academic Press, London, UK, 1979.

[9] J. Bland and D. Altman, "Multiple Significance Tests: The Bonferroni Method," *British Medical Journal*, vol. 310, 1995.