Natural Language Engineering (2019), **25**, pp. 769–772
doi:10.1017/S1351324919000536

CAMBRIDGE
UNIVERSITY PRESS

INDUSTRY WATCH

Five Tips for a Successful API

Robert Dale*

Language Technology Group, Sydney, Australia
*Corresponding author. E-mail: rdale@language-technology.com

Abstract
It's now remarkably easy to release to the world a cloud-based application programming interface (API) that provides some software function as a service. As a consequence, the cloud API space has become very densely populated, so that even if a particular API offers a service whose potential value is considerable, there are many other factors that play a role in determining whether or not that API will be commercially successful. If you're thinking about entering the API marketplace with your latest and greatest idea, this post offers some entirely subjective advice on how you might increase the chances of your offering not being lost in all the noise.

Keywords: Text analytics, APIs

Introduction

A couple of years ago, I was asked to analyse and compare a number of cloud-based text analytics APIs. One indirect result of that work was a pair of opinion pieces that appeared in this journal (Dale 2018*a*, 2018b), examining the capabilities of a number of the more well-known text analytics tools on the market. Those articles generated sufficient interest that I was encouraged to publish a much longer and more detailed in-depth analysis of 35 commercially available text analytics APIs.[a]

I give you this background to establish a bit of credibility for the focus of this piece: over the last couple of years, I have spent a significant amount of time exploring and experimenting with somewhere around 50 different APIs. A side effect of this is that I've seen a wide variety of approaches to many different aspects of making software available as a service in the cloud. Some offerings are near faultless, combining both high-quality tools with an excellent user experience; others, well, not so much.

Let's assume that the actual performance quality of a software service is a given: if a tool doesn't do what it's supposed to, then it's going to struggle to gain traction. But performance is not enough: there are many other factors that play a role in determining whether or not an API will be successful. In this piece, I draw attention to five aspects of what we might broadly think of as user experience that have impacted on my impressions of the tools I've come across.

My assessments of the importance of these characteristics fall, inevitably, in the realm of the subjective. But while it's unlikely that everyone will share my views, I think it's also unlikely that I am alone in my obsessions. So, rather than risk losing potential users who are like me, I'd like to suggest that the issues discussed below are at least worth a few minutes' consideration by anyone intending to enter the API marketplace, or, indeed, already present in that marketplace.

[a]See *Text Analytics APIs 2019: A Consumer Guide* at www.language-technology.com/apis2019.

I think it's likely that if you avoid these missteps, you'll have a greater chance of success with your enterprise.

I should emphasise that these observations have all been made in the specific context of analysing text analytics APIs, and so they should be seen in that light. But I think the points raised are relevant for many other kinds of APIs, so I hope they'll have some value beyond that narrow focus.

A final item of context setting: you may consider some of the issues pointed out below to be rather obvious. I have ample evidence, in the form of existence proofs, that they are clearly not obvious to everyone.

1. Have a professional-looking website

First impressions do matter. For most potential users, your website will be the entry point to learning about your API, so it's worth expending a reasonable amount of effort on getting this right.

Unfortunately, the websites of some APIs look like they were built in the 1980s, back in the days when it was common to handcraft HTML using emacs or vi. Both web design and the underlying technology have moved on since then. It's now remarkably easy to have a site that looks modern, well designed and aesthetically pleasing without having to code all the neat features yourself, using platforms like WordPress and Wix.

Of course, appearances can be deceptive. If you have indeed been around since the 1980s, you may now be quite established and your technology well-proven in the marketplace. But a site that looks like it was created in raw HTML doesn't encourage me to think you're using the latest technology in your API. It's more likely to make me wonder whether your tech is stuck in the 1980s too.

You don't have to be a web designer to be the proud owner of an awesome website: these days, you can access excellent design skills pretty cheaply via services like Fiverr and Upwork. Once your site is built, have an independent third-party look over your site with a critical eye, and insist that they give you an honest opinion.

And it ought to go without saying, especially if you're offering a linguisic service of some kind, but make sure you double- and triple-check your website content for spelling, grammar and style.

Of course, deceptive appearances cut both ways: I've also seen some quite stunning websites that have extremely limited product offerings behind them. But I'm willing to bet that customers looking for tools like yours are more likely to skip over sites that appear to be from a different era.

2. Make it easy to try your software

Make it very easy for me to get access to your API to try it out. Most sites these days use a completely automated sign-up mechanism where the prospective user provides their name and email address via a web form, and is immediately provided by return email with an API key that unlocks the services available. Alas, some vendors choose to introduce just a bit more friction. Worst of all of is manual processing of requests for access: please don't make me wait for a day or two for someone at your end to read their mail and grant me permissions. And please don't ask me to fill in a long form explaining how I want to use your API. Life's too short for tight shoes: I'll have given up and moved on to another vendor.

When you send an email with the API keys, make sure you provide some indication in that email of what product and website the keys are for. I know that sounds a bit odd, but if you happen to be signing up for a slew of these things all at once, it's not helpful to receive an email from a third-party subscription servicing account whose name bears no relation to the API in question, and which doesn't actually mention what product or website the key provides access to. I still have one of those lurking in the bottom of my mailbox, forever a barrier to inbox zero.

3. Provide meaningful trial access

Many APIs offer some kind of trial access, so you can experiment and explore before deciding to subscribe or otherwise buy into using the service. This is a good thing, of course. If you don't offer trial access, I strongly recommend that you do: it's very likely that your competitors do, so by not doing so, you are crippling your marketing efforts by preventing the product from speaking for itself.

Trial access models for APIs fall into two broad categories, both of which are quite common. First, you can offer the user access to a time-unlimited free subscription tier that has some limit on the number of API calls that can be made, or a limit to the specific features which are accessible (see Tip 4 below). Alternatively, you can offer the user free access to either all or a subset of the available features for a limited time period.

From a user's perspective, free tiers are by far the better option. I understand that you might not want to wear the risk of an indefinite number of lifelong users who slowly drain your resources without ever paying for them. Unfortunately, you're competing in a space where the bigger players, like Google, Microsoft and Amazon, have free tiers with high-enough usage caps that you can actually build significant applications without ever incurring a charge. The reality is that many of these technologies are effectively commodities; so it's unlikely that your now-awesome website (see Tip 1 above) will convince me you're so unique that I'll put up with a time-limited trial.

Nonetheless, if a time-limited trial is the way you feel you have to go, please at least provide a reasonable amount of time to try the product. One week (I've seen it, man) is not reasonable: more often than not, unforeseen circumstances will mean that the trial expires before the potential customer has had a chance to explore the product properly. A more sensible free trial period is 1 month.

Ideally, you will offer a free tier via your automated subscriptions mechanism. Imposing limits on the number of hits is fine, as long as its not an incredibly low number: a few thousand hits per month is reasonable in order to be able to test most APIs. 300 free hits (seen it more than once – what is it about the number 300?), on the other hand, might not be worth the trouble of writing a test harness.

4. Let me trial all your features

There's a tendency – perhaps not suprising – to mark out some features of an API as so special that you can only get access to them if you are a paying user. These are typically the API's newest and most advanced features. For example, some vendors distinguish two versions of a particular function based on different underlying technologies, with the newer and better version (at the time of writing, this is typically something makes use of deep learning) off limits to trial users.

If those features are what distinguishes your offering from others in the market, why wouldn't you let potential customers explore them as part of a trial?

It's also a good idea to keep your charging model simple. Most APIs offer a range of services (e.g., named entity recognition, sentiment analysis and text classification), and the use of each service counts as a single API hit for accounting purposes. But some vendors choose to charge a different number of units depending on the specific service called (so, e.g., classification might cost more than sentiment analysis), or even charge extra for the inclusion of specific types of results (e.g., including entity linking might cost more than simple entity recognition). If you're trying to plan a testing regime under a free trial cap, this is just a source of headaches.

5. Provide decent documentation

Many APIs are under-documented, and sometimes significantly so.

For example, if you provide a named entity recognition endpoint that assigns types to entities, it's important to indicate what the set of recognised types is. If you provide a text classification service, it's important to document the set of categories that can be assigned. Without this kind of information, it's hard to carry out sensible testing of the API.

If calls to the API can incorporate a number of parameters, document what they are and what they mean, and what the default values are. If your API returns a large collection of attributes, document them all clearly: sure, sometimes it's fairly obvious from their names, but it's always better to be explicit. If it's not clear what a returned attribute is, you might as well not bother returning it.

I'd strongly encourage the use of a formal API documentation framework. A tool like Swagger helps a lot in imposing structure and discipline around documentation, although you still need to make sure you don't fall into the trap of providing content-free descriptions of capabilities. Give complete working examples, preferably in a range of programming languages. And I mean complete: I should be able to cut and paste the examples you provide verbatim and have them run as is, perhaps modulo the insertion of an individualised API key.

It's a good idea to use version numbers for each released version of your API – preferably included in the management information that's part of any returned data from the API – so that I know when something has changed.

Don't rely on a blog post as the single source of documentation for your API. That said, blog posts can add significant support to an already well-documented API, and there are some vendors who do a really excellent job of using blog posts to demonstrate detailed use cases and the like.

Summing up

I suppose there may be legitimate reasons why the providers of some APIs don't do all the things I recommend above.

There may be other priorities that mean you don't have the time or funds to build a great website or to update the one you built a while back. Fair enough, but that may be a false economy.

Maybe your API is so special that making it hard to access, or imposing tight constraints on free usage, or limiting the capabilities I can try out, only serves to heighten the scarcity value. But chances are that, before long, someone else will take the opportunity to replicate your service and offer it on more user-friendly terms.

Maybe your target market consists of mind readers (or ex-employees) who don't need documentation or explicit examples. But I think that's likely to be a smaller target market than you might otherwise appeal to.

Ultimately, it's all about reducing friction. If you make things even just a little bit more difficult for a prospective customer than they need to be, it's not unlikely that they can find what they want elsewhere. Obvious? Perhaps; but it's still the case each of the above mistakes are committed over and over again. As I said at the beginning of this piece, I have the existence proofs. You know who you are.

One last thing: if you've tried your hand at the API business, but have decided to give it up and your API is no longer supported, please take it down. Don't leave it in a zombie-like state, with Amazon billing you the occasional 56 cents you don't even notice. And put an autoresponder on your contact emails telling me that you're not going to respond, so I don't lie awake at night wondering whether you'll ever get back to me about that parameter with the ambiguous name.

References

Dale R. (2018a). Text analytics APIs, part 1: The bigger players. *Natural Language Engineering* **24**, 317–324.
Dale R. (2018b). Text analytics APIs, part 2: The smaller players. *Natural Language Engineering* **24**, 797–803.

Cite this article: Dale R (2019). Five Tips for a Successful API. *Natural Language Engineering* **25**, 769–772. https://doi.org/10.1017/S1351324919000536