How Cute is Pikachu? Gathering and Ranking Pokémon Properties from Data with Pokémon Word Embeddings

Hämäläinen, Mika

2020

Hämäläinen , M , Alnajjar , K & Partanen , N 2020 , ' How Cute is Pikachu? Gathering and Ranking Pokémon Properties from Data with Pokémon Word Embeddings ' , Paper presented at Queer in AI , Vienna , Austria , 13/07/2020 - 17/07/2020 .

http://hdl.handle.net/10138/336370

submittedVersion

How Cute is Pikachu? Gathering and Ranking Pokémon Properties from Data with Pokémon Word Embeddings

Mika Hämäläinen, Khalid Alnajjar and Niko Partanen
Department of Digital Humanities
University of Helsinki
first.lastname@helsinki.fi

Abstract

We present different methods for obtaining descriptive properties automatically for the 151 original Pokémon. We train several different word embeddings models on a crawled Pokémon corpus, and use them to rank automatically English adjectives based on how characteristic they are to a given Pokémon. Based on our experiments, it is better to train a model with domain specific data than to use a pretrained model. Word2Vec produces less noise in the results than fastText model. Furthermore, we expand the list of properties for each Pokémon automatically. However, none of the methods is spot on and there is a considerable amount of noise in the different semantic models. Our models have been released on Zenodo.

1 Introduction

Usingknowledge-bases that contain properties typical for nouns has been in the heart of computational creativity research for a long time. Such data has proven itself useful when generating a variety of different types of creative language such as metaphors (Veale and Hao, 2007), poems (Hämäläinen, 2018) or riddles (Ritchie, 2003).

In this paper, we present a novel approach for constructing such a knowledge-base automatically for the 151 original Pokémon. Our approach is applicable in scenarios with a limited amount of data available. The resulting knowledge-base can be used in the future for generating creative language based on Pokémon such as similes and metaphors (e.g. cute as a Pikachu or confused as a Psyduck). We have made the Pokémon word embeddings models[1] freely available on Zenodo together with the Pokémon story corpus[2].

[1]Pokémon word embeddings models: https://zenodo.org/record/4554478

[2]Pokémon corpus: https://zenodo.org/record/4552785

Pokémon has been a topic of research in the past (Salter et al., 2019; Geissler et al., 2020; Vaterlaus et al., 2019). However, it has eluded any widespread NLP research interest. However, Pokémon names are surprisingly problematic for current NLP methods as we will show in this paper.

Stereotypical knowledge has been successfully extracted in the past (Veale and Hao, 2008). Their method relied on using Google search API to mine stereotypical adjective-noun relations with an "AS *adjective* AS [a/an] *NOUN*" query. However, such a method requires a lot of data in order for it to work and using such a query on a reasonably sized corpus yields hardly any results, based on our experiences.

For proper nouns, or more precisely famous characters, the simplest approach for building such a knowledge-base has been manual annotation as in the case of the Non-Official Characterization list (Veale, 2016). While, the NOC list is a valuable resource for properties for famous characters, we are looking into a more automated method for producing a similar knowledge-base for Pokémon.

There has been an automated effort for expanding the properties recorded in the NOC list (Alnajjar et al., 2017). While this method is a step towards the desired direction in the sense that it does not require the nouns to exist in a massive corpus, it still relies on mined associations between adjectival properties and a hand annotated list of properties for famous characters in order to expand them further.

In our approach, we propose a method for extracting properties for Pokémon automatically from a very small corpus. Furthermore, we use a larger Pokémon specific corpus to automatically rank the extracted properties so that a higher rank is given to the properties that are most descriptive of a given Pokémon.

Pokémon	TF-IDF	Pokémon fastText	Pre-trained fastText	Pokémon Word2Vec	Pokémon Relatedness
Parasect	back, big, dark, lower, parasite	**parasitic, poisonful, sapping, crab-like, poison**	QF, Oz, EP, XL, foe	sapping, crab-like, Polish, poison, sapped	scuttled, solar, evolved, sent, male
Omanyte	full, twisted, pokemon, original, strange	fossil, Oman, Omani, fossil-like, crab-like	JV, EP, tapu, mi, zoid	**beached, fossil, crab-like, dorsal, evolved**	fossil, caught, scald, level, prehistoric
Horsea	pokemon, original, powerful	**bubble, squirtish, high-current, splashing, swime**	QF, ray, zoid, animé, peaty	bubble, beached, high-pressured, dorsal, scald	bubble, caught, evolved, level, swimming
Arcanine	pure, true, mysterious, select, majestic	lubric, whinny, mane, canine, dismounted	EP, XL, JV, pi, glew	**whinny, earth-shaking, orange-yellow, high-pressured, scald**	back, canine, large, sent, male
Abra	original, psychic	disable, Mole, Chinglish, psychic, Minimite	Oz, ex, D., EP, Ona	**hypnotic, dinged, sapping, psychic, evolved**	psychic, teleporting, side, evolved, cast
Seaking	prominent, pokemon, original	beached, high-current, swime, dorsal, hydro	QF, JV, EP, A1, zoid	**beached, tidal, high-pressured, seismic, dorsal**	released, trapped, swimming, sent, causing
Jolteon	smallest, negative, sad, shortest, startled	mane, bristled, crackled, wagging, veed	QF, EP, XL, JV, pi	whinny, supercharged, high-pressured, pi, wagging	**evolved, electric, spiky, male, female**
Magmar	fiery, pokemon, original, smaller, intense	fire-hot, knock-on, punch, seismic, scald	foe, Oz, EP, XL, zoid	five-pointed, high-pressured, seismic, scald, hydro	**punch, fiery, sent, flame, causing**
Pidgeot	beautiful, top, wide, thick, unsuspecting	bat-wing, cawing, flappish, preened, flapped	QF, EP, XL, zoid, glew	**cawing, flapped, seismic, lightning-quick, roosting**	back, flapped, evolved, flapping, landed

Table 1: Top 5 adjectives produced by different methods for 9 randomly selected Pokémon.

2 Data and Preprocessing

In order to gather properties for each Pokémon, we look into Wikidata[3], while Wikidata does not contain descriptive properties, it provides us with unambiguous links to Giantbomb[4] entries. We use the Wikidata entry *list of Pokémon introduced in Generation I*[5] to obtain these Giantbomb links.

Giantbomb is a website listing information on video game characters. Unlike resources such a Bulbapedia[6] they provide a concise description including useful information such as characteristics and physical abilities without going too deep into the use of the Pokémon in video games. This data, however, is not structural but rather free formed textual description. This data constitutes our small Pokémon description corpus.

In order to rank Pokémon properties, we crawl a larger corpus of texts written about Pokémon. Many Wikipedia-like sources are too neutral to reveal anything meaningful about Pokémon, Pokédex entries are usually too short and non-descriptive for our needs. Subtitles form the Pokémon TV show come with their own problem of audio-visual grounding of the text. Fortunately, we found a great resource of stories authored by Pokémon fans called Fanfiction[7].

The evident problem of the resource is that many of the stories are poorly written, and that there are stories written in multiple languages. To mitigate this, we use the search functionality of the service to find stories by the query *pokemon* that are in English and have at least 10k words. This results in 8,011 fan-authored stories about Pokémon. We crawl only the stories that meet these criteria. This forms our bigger Pokémon stories corpus, which we process by doing sentence and word tokenization with NLTK (Bird et al., 2009).

3 Extracting Pokémon Properties

We experiment with multiple ways of extracting the properties for each Pokémon. In the first method, we use TF-IDF (term frequency–inverse document frequency) based method for extracting and ranking Pokémon properties on the description corpus. We compare the results of the TF-IDF method to different methods using semantic relatedness and similarity word embeddings models. For semantic relatedness we build a log-likelihood matrix of term-to-term relations based on their co-occurrences following the implementation of Meta4Meaning (Xiao et al., 2016) and for semantic similarities we utilize word2vec (Mikolov et al., 2013) and fastText (Bojanowski et al., 2016) models. We test out all the methods with generic pretrained models and with domain-specific models trained on the story corpus to see how big of a difference a domain specific corpus makes for the task of automatic extraction of properties.

We collect an initial set of adjectival properties for each Pokémon from the Pokémon description corpus by processing it using spaCy(Honnibal and Johnson, 2015) and retaining adjectives appearing in the descriptions of the Pokémon. This step yields an unranked list of few adjectival properties that are used to describe the Pokémon. However, it also includes very generic adjectives such as *original* and in some cases might find no adjectives due to very short descriptions. As an example, the properties collected for *Pikachu* included: {*electric, petite,*

[3]https://www.wikidata.org/
[4]https://www.giantbomb.com/
[5]https://www.wikidata.org/wiki/Q3245450
[6]https://bulbapedia.bulbagarden.net/
[7]https://www.fanfiction.net/

Pokémon	TF-IDF	Pokémon fastText	Pre-trained fastText	Pokémon word2Vec	Pokémon Relatedness
Drowzee	**dangerous, knowledgeable, intelligent, ruthless, twisted**	beautiful, dreamy, raw, alluring, sensuous		amusing, funny, surprising, charming, relaxed	public, versatile, despicable, specified, needed
Magnemite	light, inconspicuous, fresh, insignificant, memorable	handsome, rugged, individual		**inflexible, fixed, boring, stolid, unchanging**	soulful, grandiose, expressive, exciting, urgent
Raichu	dismayed, amazed, horrified, outraged, surprised	grandiose, funky, twisty, crazed, exciting		grandiose, funky, twisty, crazed, exciting	sturdy, potent, raw, versatile, wealthy
Beedrill	loud, dangerous, clear, deadly, slick			bitter, divisive, alive, deadly, vulgar	frustrated, disappointed, bitter, scared, shocked
Exeggcute	beautiful, creative, innovative, varied, diverse	satisfying, healthy, safe, delicious, tasty		scary, inhuman, cunning, brutal, mean	
Weezing	creative, innovative, fresh, memorable, quirky	**harmful, dangerous, deadly, slick, lethal**		**harmful, dangerous, deadly, slick, lethal**	dangerous, public, specified, needed, slick
Meowth	beautiful, professional, intelligent, expressive, versatile	**crafty, clever, funny, well-meaning, treacherous**		cunning, brutal	dominant, raised, identifying, normal, known
Ninetales	**beautiful, shiny, round, passionate, merry**	crafty, clever, funny, well-meaning, treacherous		dominant, raised, identifying, normal, known	evocative, natural, fallible, alive, feminine
Arbok	scary, dangerous, funny, intense, ruthless	fluid, dangerous, detestable, unpredictable, totalitarian		**dangerous, potent, odious, slick, carcinogenic**	dominant, feminine, identifying, damp, busted

Table 2: Expanded properties for 9 randomly selected Pokémon.

close, cute, yellow, high, . . . , first, electrical}.

Next, we investigate methods for ranking and expanding the properties of each Pokémon. The first method makes use of the TF-IDF method where we build the TF-IDF matrix from the Pokémon description corpus by treating Pokémon as documents and their descriptions as features using Scikit-learn (Pedregosa et al., 2011). The intuition here is that TF-IDF would capture the importance of each feature to Pokémon. As a result, this gives us a list of words for each Pokémon together with its strength of importance to the Pokémon. This is a very simplistic way of ranking the Pokémon properties without using the larger story corpus. Using the importance scores returned by TF-IDF to rank the properties retrieved in the previous step, we get the following ranked properties to *Pikachu*: {*lovable, onomatopoetic, prolific, stubborn, superlative, unbeknownst, . . . , -, 15th*}.

In the following steps, we rank the collected adjectival properties using semantic relatedness and similarities word embeddings models. For each method, we test out two versions, one that is pretrained on generic text such as Common Crawls and Wikipedia, and another that is trained on the Pokémon stories corpus.

We follow the approach described by (Xiao et al., 2016) to build a relatedness matrix by obtaining co-occurrences and then compute the simple log-likelihood as a measurement of relatedness between two words based on their individual frequencies and their observed and expected co-occurrences in the corpus. We use the ukWac corpus (Ferraresi et al., 2008) as the generic corpus and build two relatedness models using the generic corpus and the Pokémon stories corpus. It appears that none of the Pokémon got captured in the generic model except for two Pokémon, *Persian* and *Ditto*, which is due to the different meaning they represent in the real world. Ranking *Pikachu* properties using the domain-specific model results in: {*electric, yellow, electrical, female, quick, powerful, . . . , exclusive, maximum*}.

We use word2vec and fastText as the semantic similarity word embeddings models. We use a skip-gram model with the default hyperparameters for both fastText and word2vec. Our word embeddings method consists of having a list of properties (adjectives) the similarity of which is compared against the vector of each Pokémon by a dot product. The more similar the property is to a Pokémon, the higher it ranks. As the pretrained word2vec and fastText models, we use the models provided by (Kutuzov et al., 2017)[8] and (Mikolov et al., 2018), respectively. For our Pokémon-specific model, we utilize Gensim (Řehůřek and Sojka, 2010) to train the word2vec model and the official fastText library (Bojanowski et al., 2017) to build the fastText model from the Pokémon stories corpus.

Similarly to the generic relatedness model, Pokémon names did not appear in the pretrained word2vec model. Nonetheless, due to the fastTexts ability to use subword information during the training phase, it was able to produce semantic similarities between Pokémon and adjectival properties. Sorting *Pikachu*'s properties using the pretrained fastText and Pokémon-specific word2vec and fastText models gives:

fastText (pretrained): {*cute, chuchu, red, -, evil, yellow, japanese, . . . , tumultuous, non*},

word2vec (Pokémon): {*electric, chuchu, -, elec-*

[8]http://vectors.nlpl.eu/repository/20/3.zip

trical, quick, yellow, cute, ..., capable, prominent},

fastText (Pokémon): {*electric, chuchu, electrical, cute, yellow, close, ..., prolific, 15th*}.

In order to extract a ranked list of properties for each Pokémon from the word embedding models, we compute the similarity for each Pokémon and every single adjective in the Oxford English Dictionary (OED)[9] and sort these words (properties) based on their similarity with each Pokémon.

Furthermore, we experiment with an existing method for expanding properties for the results of each method. The property expansion is based on the data and algorithm presented by Alnajjar et al. (2017). The method takes in a list of properties and produces an extended property list by using Thesaurus Rex data (Veale and Li, 2013). We use this method to predict more properties by feeding in the top 10 adjectives produced by each model.

4 Results

Table 1 shows results for different Pokémon by the different methods. The table shows results for the word embeddings models when using adjectives from the OED. The pretrained Word2Vec model and generic relatedness model are missing from the table as they did not produce any results at all for any Pokémon. The cells in bold have the highest number of descriptive adjectives.

We can see that the pre-trained fastText model does not capture the semantics of any Pokémon at all. All in all, fastText seems to produce good adjectives in the top results, but it clearly struggles with the out-of-vocabulary adjectives. Instead of not returning a vector for them at all, and thus ignoring them, it has been designed to return vectors based on the character level similarity of the word. For this reason, *Oman* and *Omani*, words that did not occur in the training corpus, get highly associated with Omanyte, as their character distance is low. For water Pokémon, *swime*[10] gets a high score mostly due to the fact that it is close to the word *swim*.

Throughout the results, we can see that the obscurity of some of the adjectives in the OED confuses the models. Better results could be achieved if the list of adjectives was obtained from a corpus instead of a comprehensive dictionary that also records historical, obsolete and dialectal words.

It is very difficult to pick the overall best model for the task, as all of them work better for certain Pokémon than the others. We can, however, gather that word embedding models that are trained on a domain specific corpus work better than using TF-IDF to extract terms from short documents or using a pretrained model. Word2Vec seems to produce less noise than fastText.

In Table 2, we can see the resulting top 5 new properties produced by the automatic expansion of properties based on the lists for the top 10 properties produced by each method. None of the extended properties for Beedrill, Exeggcute and Raichu were descriptive enough to be highlighted as the best result. All in all, the expanded properties are very poor at describing each individual Pokémon. Based on these results, we cannot recommend using an automatic property expansion for Pokémon as it seems to favor properties typical for people. The method also failed to expand some of the properties for some models, and all of the properties for the pre-trained fastText model.

5 Conclusions

In this paper, we have presented our initial approaches in mining properties for Pokémon characters. The result look promising, although they reveal problems in the semantic representations of word embedding models, especially in pre-trained ones that belong to a different domain of text. The task of automatically extracting meaningful properties is far from trivial and calls for more future work. Nonetheless, our approach is a step away from expert annotated data into a fully automatic methodology.

The journey has just begun, so in the future different experiments could be conducted in terms of what kind of adjectives are used to query the word embedding models for each Pokémon. Also, a hybrid approach could be taken to combine the strengths of each individual model; the more models point towards a certain property, the more likely it is to be a descriptive one of a given Pokémon.

Based on our research, we can conclude that the pretrained models do not work with Pokémon at all. Clearly, Pokémon itself is by no means so deviant a phenomenon that it could not be modeled with word embeddings. The problem we can see is part of a wider phenomenon that has received a little attention in the field of NLP. If pretrained models,

[9] https://www.oed.com/

[10] OED: Used vaguely (like the noun) in Destr. Troy = giddy, dazed, and (actively) stunning.

which are constantly used in various NLP studies, are not able to describe Pokémon, what other phenomena might they describe equally poorly? In general, our discipline does not pay very much attention to how well computational models work when applied to a completely new context.

The embeddings trained in this paper may be useful in a variety of different computational creativity tasks relating to Pokémon. Therefore we have released the models and the code freely on Zenodo (links on the first page of this paper).

References

Khalid Alnajjar, Mika Hämäläinen, Hanyang Chen, and Hannu Toivonen. 2017. Expanding and weighting stereotypical properties of human characters for linguistic creativity. In *ICCC*, pages 25–32.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*, 1st edition. O'Reilly Media, Inc.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Adriano Ferraresi, Eros Zanchetta, Marco Baroni, and Silvia Bernardini. 2008. Introducing and evaluating ukwac, a very large web-derived corpus of english. In *Proceedings of the 4th Web as Corpus Workshop (WAC-4) Can we beat Google*, pages 47–54.

Dominique Geissler, Elisa Nguyen, Daphne Theodorakopoulos, and Lorenzo Gatti. 2020. Pokérator-unveil your inner pokémon. In *Proceedings of the Eleventh International Conference on Computational Creativity*.

Mika Hämäläinen. 2018. Harnessing nlg to create finnish poetry automatically. In *International Conference on Computational Creativity*, pages 9–15. Association for Computational Creativity (ACC).

Matthew Honnibal and Mark Johnson. 2015. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal. Association for Computational Linguistics.

Andrei Kutuzov, Murhaf Fares, Stephan Oepen, and Erik Velldal. 2017. Word vectors, reuse, and replicability: Towards a community repository of large-text resources. In *Proceedings of the 58th Conference on Simulation and Modelling*, pages 271–276. Linköping University Electronic Press.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA.

Graeme Ritchie. 2003. The jape riddle generator: technical specification. *Institute for Communicating and Collaborative Systems*.

Anastasia Salter, Mel Stanfill, and Anne Sullivan. 2019. But does pikachu love you? reproductive labor in casual and hardcore games. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, New York, NY, USA. Association for Computing Machinery.

J Mitchell Vaterlaus, Kala Frantz, and Tracey Robecker. 2019. "reliving my childhood dream of being a pokémon trainer": An exploratory study of college student uses and gratifications related to pokémon go. *International Journal of Human–Computer Interaction*, 35(7):596–604.

Tony Veale. 2016. Round up the usual suspects: Knowledge-based metaphor generation. In *Proceedings of the Fourth Workshop on Metaphor in NLP*, pages 34–41.

Tony Veale and Yanfen Hao. 2007. Comprehending and generating apt metaphors: a web-driven, case-based approach to figurative language. In *AAAI*, volume 2007, pages 1471–1476.

Tony Veale and Yanfen Hao. 2008. Enriching wordnet with folk knowledge and stereotypes. In *Proceedings of the 4th Global WordNet Conference, Szeged, Hungary*.

Tony Veale and Guofu Li. 2013. Creating similarity: Lateral thinking for vertical similarity judgments. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 660–670.

Ping Xiao, Khalid Alnajjar, Mark Granroth-Wilding, Kat Agres, and Hannu Toivonen. 2016. Meta4meaning: Automatic metaphor interpretation using corpus-derived word associations. In *Proceedings of the 7th International Conference on Computational Creativity (ICCC). Paris, France*.