How to Make a Frenemy: Multitape FSTs for Portmanteau Generation

Aliya Deri and Kevin Knight
Information Sciences Institute
Department of Computer Science
University of Southern California
{aderi, knight}@isi.edu

Abstract

A portmanteau is a type of compound word that fuses the sounds and meanings of two component words; for example, "frenemy" (friend + enemy) or "smog" (smoke + fog). We develop a system, including a novel multitape FST, that takes an input of two words and outputs possible portmanteaux. Our system is trained on a list of known portmanteaux and their component words, and achieves 45% exact matches in cross-validated experiments.

W^1	W^2	PM
affluence	influenza	affluenza
anecdote	data	anecdata
chill	relax	chillax
flavor	favorite	flavorite
guess	estimate	guesstimate
jogging	juggling	joggling
sheep	people	sheeple
spanish	english	spanglish
zeitgeist	ghost	zeitghost

Table 1: Valid component words and portmanteaux.

1 Introduction

Portmanteaux are new words that fuse both the sounds and meanings of their component words. Innovative and entertaining, they are ubiquitous in advertising, social media, and newspapers (Figure 1). Some, like "frenemy" (friend + enemy), "brunch" (breakfast + lunch), and "smog" (smoke + fog), express such unique concepts that they permanently enter the English lexicon.

Portmanteau generation, while seemingly trivial for humans, is actually a combination of two complex natural language processing tasks: (1) choosing component words that are both semantically and phonetically compatible, and (2) blending those words into the final portmanteau. An end-to-end system that is able to generate novel portmanteaux with minimal human intervention would be not only a useful tool in areas like advertising and journalism, but also a notable achievement in creative NLP.


DEPT. OF HOSPITALITY | JANUARY 27, 2014 ISSUE

STAYCATION

Figure 1: A *New Yorker* headline portmanteau.

Due to the complexity of both component word selection and blending, previous portmanteau generation systems have several limitations. The Nehovah system (Smith et al., 2014) combines words only at exact grapheme matches, making the generation of more complex phonetic blends like "frenemy" or "brunch" impossible. Özbal and Strappavara (2012) blend words phonetically and allow inexact matches but rely on encoded human knowledge, such as sets of similar phonemes and semantically related words. Both systems are rule-based, rather than data-driven, and do not train or test their systems with real-world portmanteaux.

In contrast to these approaches, this paper presents a data-driven model that accomplishes (2) by blending two given words into a portmanteau. That is, with an input of "friend" and "enemy," we want to generate "frenemy."

Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL, pages 206–210,
Denver, Colorado, May 31 – June 5, 2015.

$F_1\ R_1\ EH_3\ N_3\ D_4$ $\quad$ $T_1\ OW_1\ F_1\ UW_3$

$EH_3\ N_3\ AH_5\ M_5\ IY_5$ $\quad$ $T_2\ ER_3\ K_5\ IY_5$

Figure 2: Derivations for friend + enemy → "frenemy" and tofu + turkey → "tofurkey." Subscripts indicate the step applied to each phoneme.

We take a statistical modeling approach to portmanteau generation, using training examples (Table 1) to learn weights for a cascade of finite state machines. To handle the 2-input, 1-output problem inherent in the task, we implement a multitape FST.

This work's contributions can be summarized as:

- a portmanteau generation model, trained in an unsupervised manner on unaligned portmanteaux and component words,
- the novel use of a multitape FST for a 2-input, 1-output problem, and
- the release of our training data.[1]

2 Definition of a portmanteau

In this work, a portmanteau PM and its pronunciation $\mathrm{PM_{pron}}$ have the following constraints:

- PM has exactly 2 component words W^1 and W^2, with pronunciations $\mathrm{W}^1_{\mathrm{pron}}$ and $\mathrm{W}^2_{\mathrm{pron}}$.
- All of PM's letters are in W^1 and W^2, and all phonemes in $\mathrm{PM_{pron}}$ are in $\mathrm{W}^1_{\mathrm{pron}}$ and $\mathrm{W}^2_{\mathrm{pron}}$.
- All pronunciations use the Arpabet symbol set.
- Portmanteau building occurs at the phoneme level. $\mathrm{PM_{pron}}$ is built through the following steps (further illustrated in Figure 2):

1. 0+ phonemes from $\mathrm{W}^1_{\mathrm{pron}}$ are output.
2. 0+ phonemes from $\mathrm{W}^2_{\mathrm{pron}}$ are deleted.
3. 1+ phonemes from $\mathrm{W}^1_{\mathrm{pron}}$ are aligned with an equal number of phonemes from $\mathrm{W}^2_{\mathrm{pron}}$.
   For each aligned pair of phonemes (x, y), either x or y is output.
4. 0+ phonemes from $\mathrm{W}^1_{\mathrm{pron}}$ are deleted, until the end of $\mathrm{W}^1_{\mathrm{pron}}$.
5. 0+ phonemes from $\mathrm{W}^2_{\mathrm{pron}}$ are output, until the end of $\mathrm{W}^2_{\mathrm{pron}}$.

[1] Available at both authors' websites.

3 Multitape FST model

Finite state machines (FSMs) are powerful tools in NLP and are frequently used in tasks like machine transliteration and pronunciation. Toolkits like Carmel and OpenFST allow rapid implementations of complex FSM cascades, machine learning algorithms, and n-best lists.

Both toolkits implement two types of FSMs: finite state acceptors (FSAs) and finite state transducers (FSTs), and their weighted counterparts (wFSAs and wFSTs). An FSA has one input tape; an FST has one input and one output tape.

What if we want a one input and two output tapes for an FST? Three input tapes for an FSA? Although infrequently explored in NLP research, these "multitape" machines are valid FSMs.

In the case of converting $\{\mathrm{W}^1_{\mathrm{pron}}, \mathrm{W}^2_{\mathrm{pron}}\}$ to $\mathrm{PM_{pron}}$, an interleaved reading of two tapes would be impossible with a traditional FST. Instead, we model the problem with a 2-input, 1-output FST (Figure 3). Edges are labeled $x : y : z$ to indicate input tapes $\mathrm{W}^1_{\mathrm{pron}}$ and $\mathrm{W}^2_{\mathrm{pron}}$ and output tape $\mathrm{PM_{pron}}$, respectively.

4 FSM Cascade

We include the multitape model as part of an FSM cascade that converts W^1 and W^2 to PM (Figure 4).

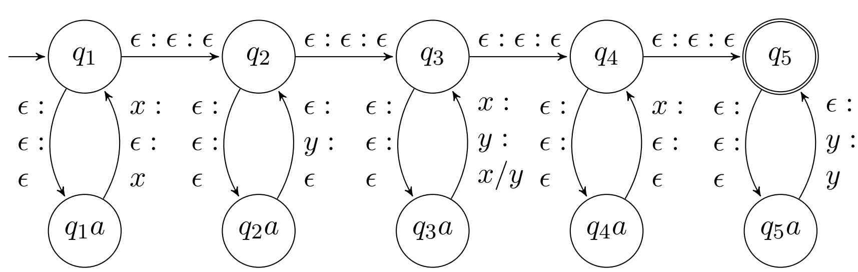

Figure 3: A 2- input, 1-output wFST for portmanteau pronunciation generation.

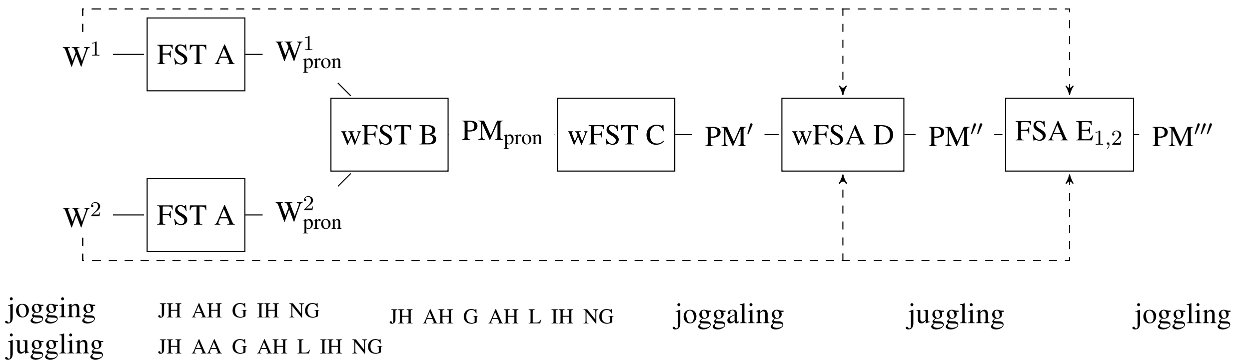

Figure 4: The FSM cascade for converting W^1 and W^2 into a PM, and an illustrative example.

phonemes			$P(x, y \to z)$		
x	y	z	cond.	joint	mixed
AA	AA	AA	1.000	0.017	1.000
AH	ER	AH	0.424	0.007	0.445
AH	ER	ER	0.576	0.009	0.555
P	B	P	0.972	0.002	1.000
P	B	B	0.028	N/A	N/A
Z	SH	SH	1.000	N/A	N/A
JH	AO	JH	1.000	N/A	N/A

Table 2: Sample learned phoneme alignment probabilities for each method.

step k	description	$P(k)$
1	$\mathrm{W}^1_{\mathrm{pron}}$ keep	0.68
2	$\mathrm{W}^2_{\mathrm{pron}}$ delete	0.55
3	align	0.74
4	$\mathrm{W}^1_{\mathrm{pron}}$ delete	0.64
5	$\mathrm{W}^2_{\mathrm{pron}}$ keep	0.76

Table 3: Learned step probabilities. The probabilities of keeping and aligning are higher than those of deleting, showing a tendency to preserve the component words.

We first generate the pronunciations of W^1 and W^2 with FST A, which functions as a simple look-up from the CMU Pronouncing Dictionary (Weide, 1998).

Next, wFST B, the multitape wFST from Figure 3, translates $\mathrm{W}^1_{\mathrm{pron}}$ and $\mathrm{W}^2_{\mathrm{pron}}$ into $\mathrm{PM}_{\mathrm{pron}}$. wFST C, built from aligned graphemes and phonemes from the CMU Pronunciation Dictionary (Galescu and Allen, 2001), spells $\mathrm{PM}_{\mathrm{pron}}$ as PM′.

To improve PM′, we now use three FSAs built from W^1 and W^2. The first, wFSA D, is a smoothed "mini language model" which strongly prefers letter trigrams from W^1 and W^2. The second and third, FSA E_1 and FSA E_2, accept all inputs except W^1 and W^2.

5 Data

We obtained examples of portmanteaux and component words from Wikipedia and Wiktionary lists (Wikipedia, 2013; Wiktionary, 2013). We reject any that do not satisfy our constraints–for example, portmanteaux with three component words ("turkey" + "duck" + "chicken" → "turducken") or without any overlap ("arpa" + "net" → "arpanet"). From 571 examples, this yields 401 {W^1, W^2, PM} triples.

We also use manual annotations of $\mathrm{PM}_{\mathrm{pron}}$ for learning the multitape wFST B weights and for mid-cascade evaluation.

We randomly split the data for 10-fold cross-validation. For each iteration, 8 folds are used for training data, 1 for dev, and 1 for test. Training data is used to learn wFST B weights (Section 6) and dev data is used to learn reranking weights (Section 7).

6 Training

FST A is unweighted and wFST C is pretrained. wFSA D and FSA $\mathrm{E}_{1,2}$ are built at runtime.

We only need to learn wFST B weights, which we can reduce to weights on transitions $q_k \to q_k a$ and $q_3 a \to q_3$ from Figure 3. The weights $q_k \to q_k a$ represent the probability of each step, or $P(k)$. The weights $q_3 a \to q_3$ represent the probability of generating phoneme z from input phonemes x and y, or $P(x, y \to z)$.

model	% exact		avg. dist.		% 1k-best	
	dev	test	dev	test	dev	test
cond	28.9	29.9	1.6	1.6	92.0	**91.2**
joint	**44.6**	**44.6**	**1.5**	**1.5**	91.0	89.7
mixed	31.9	33.4	1.6	1.5	**92.8**	91.0
rerank	**51.4**	**50.6**	**1.2**	**1.3**	**93.1**	**91.5**

Table 4: PM_{pron} results pre- and post-reranking.

PM	% exact	avg. dist.	% 1k-best
PM'	12.03	5.31	42.35
PM''	42.14	1.80	58.10
PM'''	**45.39**	**1.59**	**61.35**

Table 5: PM results on cross-validated test data.

We use expectation maximization (EM) to learn these weights from our unaligned input and output, $\{W^1_{pron}, W^2_{pron}\}$ and PM_{pron}. We use three different methods of normalizing fractional counts. The learned phoneme alignment probabilities $P(x, y \rightarrow z)$ (Table 2) vary across these methods, but the learned step probabilities $P(k)$ (Table 3) do not.

6.1 Conditional Alignment

Our first learning method models phoneme alignment $P(x, y \rightarrow z)$ conditionally, as $P(z|x, y)$. Since $P(z|x, y)$ tends to be larger than step probabilities $P(k)$, the model prefers to align phonemes when possible, rather than keep or delete them separately. This creates longer alignment regions.

Additionally, during training a potential alignment $P(x|x, y)$ can compete only with its pair $P(y|x, y)$, making it more difficult to zero out an alignment's probability. The conditional method therefore also learns more potential alignments between phonemes.

6.2 Joint Alignment

Our second learning method models $P(x, y \rightarrow z)$ jointly, as $P(z, x, y)$. Since $P(z, x, y)$ is relatively low compared to the step probabilities, this method prefers very short alignments–the reverse of the effect seen in the conditional method.

However, the model can also zero out the probabilities of unlikely aligments, so overall it learns fewer possible alignments between phonemes.

W^1	W^2	gold PM	hyp. PM
affluence	influenza	affluenza	affluenza
architecture	ecology	arcology	architecology
chill	relax	chillax	chilax
friend	enemy	frenemy	frienemy
japan	english	japlish	japanglish
jeans	shorts	jorts	js
jogging	juggling	joggling	joggling
man	purse	murse	mman
tofu	turkey	tofurkey	tofurkey
zeitgeist	ghost	zeitghost	zeitghost

Table 6: Component words and gold and hypothesis PMs.

6.3 Mixed Alignment

Our third learning method initializes alignment probabilities with the joint method, then normalizes them so that $P(x|x, y)$ and $P(y|x, y)$ sum to 1. This "mixed" method, like the joint method, is more conservative in learning phoneme alignments. However, like the conditional method, it has high alignment probabilities and prefers longer alignments.

7 Model Combination and Reranking

Using the methods from sections 6.1, 6.2, and 6.3, we train three models and produce three different 1000-best lists of PM_{pron} candidates for dev data. We combine these three lists into a single one, and compute the following features for each candidate: model scores, PM_{pron} length, percentage of W^1_{pron} or W^2_{pron} in PM_{pron}, and percentage of PM_{pron} in W^1_{pron} or W^2_{pron}. We also include a binary feature for whether PM_{pron} matches W^1_{pron} or W^2_{pron}.

We then compute feature weights using the averaged perceptron algorithm (Zhou et al., 2006), and use them to rerank the candidate list, for both dev and test data. We combine the reranked PM_{pron} lists to generate wFST C's input.

8 Evaluation

We evaluate our model's generation of PM_{pron} pre- and post-reranking against our manually annotated PM_{pron}. We also compare PM', PM'', and PM'''. For both PM_{pron} and PM, we use three metrics:

- percent of 1-best results that are exact matches,
- average Levenshtein edit distance of 1-bests, and
- percent of 1000-best lists with an exact match.

9 Results and Discussion

We first evaluate the model at $\mathrm{PM}_{\mathrm{pron}}$. Table 4 shows that, despite less than 50% exact matches, over 90% of the 1000-best lists contain the correct pronunciation. This motivates our model combination and reranking, which increase exact matches to over 50%.

Next, we evaluate PM (Table 5). A component word mini-LM dramatically improves PM'' compared to PM'. Filtering out component words provides additional gain, to 45% exact matches.

In comparison, a baseline that merges $\mathrm{W}^1_{\mathrm{pron}}$ and $\mathrm{W}^2_{\mathrm{pron}}$ at the first shared phoneme achieves 33% exact matches for $\mathrm{PM}_{\mathrm{pron}}$ and 25% for PM.

Table 6 provides examples of system output. Perfect outputs include "affluenza," "joggling," "tofurkey," and "zeitghost." For others, like "chilax" and "frienemy," the discrepancy is negligible and the hypothesis PM could be considered a correct alternate output. Some hypotheses, like "architecology" and "japanglish," might even be considered superior to their gold counterparts. However, some errors, like "js" and "mman," are clearly unacceptable system outputs.

10 Conclusion

We implement a data-driven system that generates portmanteaux from component words. To accomplish this, we use an FSM cascade, including a novel 2-input, 1-output multitape FST, and train it on existing portmanteaux. In cross-validated experiments, we achieve 45% exact matches and an average Levenshtein edit distance of 1.59.

In addition to improving this model, we are interested in developing systems that can select component words for portmanteaux and reconstruct component words from portmanteaux. We also plan to research other applications for multi-input/output models.

11 Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments, as well as our colleagues Qing Dou, Tomer Levinboim, Jonathan May, and Ashish Vaswani for their advice. This work was supported in part by DARPA contract FA-8750-13-2-0045.

References

Lucian Galescu and James F Allen. 2001. Bi-directional conversion between graphemes and phonemes using a joint n-gram model. In *4th ISCA Tutorial and Research Workshop (ITRW) on Speech Synthesis*.

Gözde Özbal and Carlo Strapparava. 2012. A computational approach to the automation of creative naming. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 703–711. Association for Computational Linguistics.

Michael R Smith, Ryan S Hintze, and Dan Ventura. 2014. Nehovah: A neologism creator nomen ipsum. In *Proceedings of the International Conference on Computational Creativity*, pages 173–181. ICCC.

Robert Weide. 1998. The CMU pronunciation dictionary, release 0.6.

Wikipedia. 2013. List of portmanteaus. `http://en.wikipedia.org/w/index.php?title=List_of_portmanteaus&oldid=578952494`. [Online; accessed 01-November-2013].

Wiktionary. 2013. Appendix:list of portmanteaux. `http://en.wiktionary.org/w/index.php?title=Appendix:List_of_portmanteaux&oldid=23685729`. [Online; accessed 02-November-2013].

Zhengyu Zhou, Jianfeng Gao, Frank K. Soong, and Helen Meng. 2006. A comparative study of discriminative methods for reranking LVCSR n-best hypotheses in domain adaptation and generalization. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing, ICASSP 2006, Toulouse, France*, pages 141–144.