Inspire at SemEval-2016 Task 2: Interpretable Semantic Textual Similarity Alignment based on Answer Set Programming

Mishal Kazmi
Sabanci University
Istanbul, Turkey
mishalkazmi@sabanciuniv.edu

Peter Schüller
Marmara University
Istanbul, Turkey
peter.schuller@marmara.edu.tr

Abstract

In this paper we present our system developed for the SemEval 2016 Task 2 - Interpretable Semantic Textual Similarity along with the results obtained for our submitted runs. Our system participated in the subtasks predicting chunk similarity alignments for gold chunks as well as for predicted chunks. The Inspire system extends the basic ideas from last years participant NeRoSim, however we realize the rules in logic programming and obtain the result with an Answer Set Solver. To prepare the input for the logic program, we use the PunktTokenizer, Word2Vec, and WordNet APIs of NLTK, and the POS- and NER-taggers from Stanford CoreNLP. For chunking we use a joint POS-tagger and dependency parser and based on that determine chunks with an Answer Set Program. Our system ranked third place overall and first place in the Headlines gold chunk subtask.

1 Introduction

Semantic Textual Similarity (STS), refers to the degree of semantic equivalence between a pair of texts. This helps in explaining how some texts are related or unrelated. In Interpretable STS (iSTS) systems, further explanation is provided as to why the two texts are related or unrelated. Finding these detailed explanations helps in gathering a meaningful representation of their similarities.

The competition at SemEval 2016 was run on three different datasets: Headlines, Images and Student-Answers. Each dataset included two files containing pairs of sentences and two files containing pairs of gold-chunked sentences. Either the gold chunks provided by the organizers or chunks obtained from the given texts would be used as input to the system. The expected outputs of the system are m:n chunk-chunk alignments, corresponding similarity scores between 0 (unrelated) and 5(equivalent), and a label indicating the type of semantic relation. Possible semantic relations are EQUI (semantically equal), OPPO (opposite), SPE1/SPE2 (chunk 1/2 is more specific than chunk 2/1), SIMI (similar but none of the relations above), REL (related but none of the relations above), and NOALI (not aligned, e.g., punctuation). All relations shall be considered in the given context. For details see the companion paper (Agirre et al., 2016).

Similarity alignment in the Inspire system is based on ideas of previous year's NeRoSim (Banjade et al., 2015) entry, however we reimplemented the system and realize the rule engine in Answer Set Programming (ASP) (Lifschitz, 2008; Brewka et al., 2011) which gives us flexibility for reordering rules or applying them in parallel. To account for differences in datasets we detect the type of input with a Naive Bayes classifier and use different parameter sets for each of the datasets.

Chunking in the Inspire system is based on a joint POS-tagger and dependency parser (Bohnet et al., 2013) and an ASP program that determines chunk boundaries.

Proceedings of SemEval-2016, pages 693–699,
San Diego, California, June 16-17, 2016.

In Section 2 we give preliminaries of ASP, Section 3 describes how we represented semantic alignment in ASP, in Section 5 we explain parameter sets used for different datasets, Section 4 briefly outlines our chunking approach, and we give evaluation results in Section 6.

2 Answer Set Programming (ASP)

Answer Set Programming (ASP) (Lifschitz, 2008; Brewka et al., 2011) is a logic programming paradigm based on rules of the form:

$$a \leftarrow b_1,\ \ldots, b_m, \mathbf{not}\, b_{m+1}, \ldots, \mathbf{not}\, b_n$$

where a, b_i are atoms from a first-order language, a is the head and $b_1, \ldots, \mathbf{not}\, b_n$ is the body of the rule, and **not** is negation as failure. Variables start with capital letters, facts (rules without body condition) are written as 'a.' instead of '$a \leftarrow$'. Intuitively a is true if all positive body atoms are true and no negative body atom is true. ASP is declarative, i.e., the order of rules and the order of body atoms in rules does not matter (different from Prolog), a popular usage pattern of ASP is to create three program modules called Generate, Define, and Test, which (i) generate a space of potential solutions, (ii) define auxiliary concepts, and (iii) eliminate invalid solutions using constraints, respectively. ASP allows encoding of nondeterministic polynomial time (NP)-hard problems, solver tools usually first instantiate the given program and then find solutions of the variable-free theory using adapted Satisfiability (SAT) solvers. Apart from normal atoms and rules, ASP supports aggregates and other constructions, for details we refer to the ASP-Core-2 standard (Calimeri et al., 2012). In this work we use the ASP solvers Gringo and Clasp (Gebser et al., 2011) as a Python library.

3 Alignment based on ASP

For each sentence pair, we align chunks in the following architecture:

- Chunked input sentence pairs are preprocessed (POS, NER, Word2Vec, WordNet) and represented as a set of ASP facts (3.1).
- A generic set of rules represents how alignments can be defined and changed (3.2).
- We represent alignments based on the description of the NeRoSim engine (3.3).
- We evaluate the above components in an ASP solver, obtaining answers sets containing a representation of alignments. From this we create the system output file (3.4).

3.1 Preprocessing

In this step we create facts that represent the input, including lemmatization via NLTK (Bird, 2006), POS- and NER-tagging from Stanford CoreNLP (Manning et al., 2014), lookups in WordNet (Miller, 1995), and similarity values obtained using Word2Vec (Mikolov et al., 2013).

We explain the input representation of the following sentence pair:

[A tan puppy] [being petted] [.]
[A tan puppy] [being held and petted] [.]

We first represent sentences, chunks, and words in chunks with POS-tags, NER-tags, and lowercase versions of words as follows:

$sentence(1).$
$chunk(sc(1,0)).$
$chunk(sc(1,1)).$
$chunk(sc(1,2)).$
$mword(cw(sc(1,0),1),\text{"}A\text{"},\text{"}a\text{"},\text{"}DT\text{"},\text{"}O\text{"}).$
$mword(cw(sc(1,0),2),\text{"}tan\text{"},$
$\text{"}tan\text{"},\text{"}NN\text{"},\text{"}O\text{"}).$
$mword(cw(sc(1,0),3),\text{"}puppy\text{"},$
$\text{"}puppy\text{"},\text{"}NN\text{"},\text{"}O\text{"}).$
...
$sentence(2).$
$chunk(sc(2,0)).$
...

Intuitively a sentence ID is an integer, a chunk ID is a composite $sc(sentenceID, chunkIdx)$, and a word ID is a composite $cw(chunkID, wordIdx)$.

We detect punctuation, cardinal numbers, and dates/times using regular expressions and represent this as facts $punct(\cdot)$, $cardinalnumber(\cdot)$, and $datetime(\cdot)$, resp., e.g.,

$$punct(cw(sc(1,2),0)).$$

We lookup synonyms, hypernyms, and antonyms in WordNet and add the respective facts.

$$synonym(\text{"}a\text{"}, \text{"}a\text{"}).$$
$$synonym(\text{"}a\text{"}, \text{"}vitamin\ a\text{"}).$$
$$synonym(\text{"}tan\text{"}, \text{"}burn\text{"}).$$
$$hypernym(\text{"}puppy\text{"}, \text{"}dog\text{"}).$$
$$hypernym(\text{"}puppy\text{"}, \text{"}domestic\_dog\text{"}).$$
$$hypernym(\text{"}puppy\text{"}, \text{"}canis\_familiaris\text{"}).$$
$$\ldots$$

We use distributional similarity with the Word2Vec tool (Mikolov et al., 2013) trained on the One Billion Word[1] benchmark (Chelba et al., 2014) with SkipGram context representation, window size 10, vector dimension 200, and pruning below frequency 50. Word-word similarity $sim(v, w)$ is computed using cosine similarity from scikit-learn (Pedregosa et al., 2011), between vectors of words v and w. We compute chunk-chunk similarity as

$$\frac{best(1,2) + best(2,1)}{2\min(n_1, n_2)}, \text{ where} \quad (1)$$

$$best(x, y) = \sum_{i=1}^{n_x} \max_{j=1}^{n_y} sim(w_i^x, w_j^y)$$

with n_x the number of words in chunk x and w_b^x the word of index b in chunk x. This is based on (Banjade et al., 2015, Section 2.2.2).

We represent chunk-chunk similarity as atoms $chunksimilarity(chunk1Id, chunk2Id, S)$. In our example this creates, e.g., the following facts.

$$chunksimilarity(sc(1,0), sc(2,0), 101).$$
$$chunksimilarity(sc(1,0), sc(2,1), 22).$$
$$chunksimilarity(sc(1,0), sc(2,2), 2).$$
$$chunksimilarity(sc(1,1), sc(2,0), 34).$$
$$\ldots$$

Note that similarity can be above 100 due to dividing by the shorter chunk length.

3.2 Rule Engine

To make POS, NER, word, and lowercase words more accessible, we project them to new facts with the following rules

$$word(Id, W) \leftarrow mword(Id, W, \_, \_, \_).$$
$$lword(Id, L) \leftarrow mword(Id, \_, L, \_, \_).$$
$$pos(Id, P) \leftarrow mword(Id, \_, \_, P, \_).$$
$$ner(Id, N) \leftarrow mword(Id, \_, \_, \_, N).$$

where the arguments of *mword* are word ID, word, lowercase word, POS and NER tag.

Variables of the form '_' are anonymous, intuitively these values are projected away before applying the rule.

We interpret POS and NER tags, and mark nouns, verbs, contentwords, proper nouns, cardinal numbers and locations based on tags from the Penn Treebank (Marcus et al., 1993).

$$noun(Id) \leftarrow pos(Id, \text{"}NNS\text{"}).$$
$$noun(Id) \leftarrow pos(Id, \text{"}NNP\text{"}).$$
$$verb(Id) \leftarrow pos(Id, \text{"}VB\text{"}).$$
$$verb(Id) \leftarrow pos(Id, \text{"}VBD\text{"}).$$
$$\ldots$$
$$location(Id) \leftarrow ner(Id, \text{"}LOCATION\text{"}).$$

We ensure symmetry of chunk similarity, synonyms and antonyms, and transitivity of the synonym relation.

$$chunksimilarity(C_2, C_1, S) \leftarrow chunksimilarity(C_1, C_2, S).$$
$$synonym(W, W') \leftarrow synonym(W', W).$$
$$antonym(W, W') \leftarrow antonym(W', W).$$
$$synonym(W_1, W_3) \leftarrow synonym(W_1, W_2), synonym(W_2, W_3).$$

We define pairs of chunks that can be matched together with their sentence indices. This is useful because this way we can define conditions on potential alignments without specifying the direction of alignment (1 to 2 vs. 2 to 1).

$$chpair(S_1, S_2, sc(S_1, C_1), sc(S_2, C_2)) \leftarrow chunk(sc(S_1, C_1)), chunk(sc(S_2, C_2)), S_1 \neq S_2.$$

We represent alignments as follows:

(i) alignment happens in steps that have an order defined by atoms $nextStep(S, S')$ which indicates that S happens before S',

[1] http://www.statmt.org/lm-benchmark/

(ii) a chunk can be aligned to one or more chunks only within one step, afterwards alignment cannot be changed,

(iii) program modules can define atoms of form $chalign(C_1, R, Sc, C_2, St)$ which indicates that chunks C_1 and C_2 should be aligned with label R (e.g., "EQUI") and score Sc (e.g., 5) in step St.

Defining an alignment is possible if the chunks are not yet aligned, it marks both involved chunks as aligned, and they stay aligned.

$$aligned(C, S') \leftarrow \textbf{not } aligned(C, S), \\ chalign(C, \_, \_, \_, S), nextStep(S, S').$$
$$aligned(C, S') \leftarrow \textbf{not } aligned(C, S), \\ chalign(\_, \_, \_, C, S), nextStep(S, S').$$
$$aligned(C, S') \leftarrow aligned(C, S), nextStep(S, S').$$

Final alignments (that are interpreted by Python) are represented using predicate *final*, these atoms include raw chunk similarity (for experimenting with other ways to define the similarity score). Moreover only steps that are used (in $nextStep(\cdot, \cdot)$) are included.

$$usedStep(S) \leftarrow nextStep(S, \_).$$
$$usedStep(S') \leftarrow nextStep(\_, S').$$
$$final(C_1, Rel, Score, C_2, S, Simi) \leftarrow \\ chalign(C_1, Rel, Score, C_2, S), \\ \textbf{not } aligned(C_1, S), \textbf{not } aligned(C_2, S), \\ usedStep(S), chunksimilarity(C_1, C_2, Simi).$$

This system gives us flexibility for configuring the usage and application of the order of rules by defining *nextStep* accordingly (see Section 5).

3.3 NeRoSim Rules

All that remains is to realize the NeRoSim rules, labeled with individual steps, and add them to the program.

In the following we give an example of how we realize NeRoSim's condition C_1 (one chunk has a conjunction and other does not).

$$cond_1(C_1, C_2) \leftarrow chpair(\_, \_, C_1, C_2), \\ \#count\{W_1 : conjunction(cw(C_1, W_1))\} = 0, \\ \#count\{W_2 : conjunction(cw(C_2, W_2))\} \geq 1.$$

Intuitively, the $\#count$ aggregates become true if the appropriate number of atoms in the set becomes true.

As a second example, our adaptation of rule SP_1 for SPE1/SPE2 alignments defines $sp1(C_1, C_2)$ if $cond_1$ holds between C_2 and C_1 and if C_1 contains all content words of C_2. Note that $contentword\_subset(A, B)$ is defined separately for chunks A, B if B contains all content words of A.

$$sp1(C_1, C_2) \leftarrow chpair(\_, \_, C_1, C_2), \\ cond_1(C_2, C_1), contentword\_subset(C_2, C_1).$$

We use *sp1* in our stepwise alignment engine by defining *chalign* with SPE1 and SPE2 according to which *sentence* is more specific.

$$chalign(C_1, \text{"}SPE1\text{"}, 4, C_2, sp1) \leftarrow \\ chpair(1, 2, C_1, C_2), sp1(C_1, C_2).$$
$$chalign(C_1, \text{"}SPE2\text{"}, 4, C_2, sp1) \leftarrow \\ chpair(1, 2, C_1, C_2), sp1(C_2, C_1).$$

For reasons of space we are unable to list all NeRoSim rules and their ASP realization. For the description of rules NOALIC, EQUI(1–5), OPPO, SPE1/2(1–3), SIMI(1–5), REL, we refer to (Banjade et al., 2015).

The full ASP code is publicly available.[2]

3.4 Interpretation of Answer Sets

After the evaluation of the above rules with the facts that describe the input sentences (Section 3.1) the solver returns a set of answer sets (in our case a single answer set). This answer set contains all true atoms and we are interested only in the *final* predicates.

$$word(cw(sc(1, 1), 4), \text{"}being\text{"}).$$
$$word(cw(sc(1, 1), 5), \text{"}petted\text{"}).$$
$$word(cw(sc(2, 1), 4), \text{"}being\text{"}).$$
$$word(cw(sc(2, 1), 5), \text{"}held\text{"}).$$
$$word(cw(sc(2, 1), 6), \text{"}and\text{"}).$$
$$\ldots$$
$$final(sc(1, 0), \text{"}EQUI\text{"}, 5, sc(2, 0), equi1, 101).$$

[2] https://bitbucket.org/snippets/knowlp/yrjqr

final(*sc*(*1*, *1*), "*SPE2*", *4*, *sc*(*2*, *1*), *sp1*, *106*).

From these predicates we create the required output which is a single continous line of the following form:

```
4 5 6 <==> 4 5 6 7 // SPE2 // 4 //
  being petted <==> being held and petted
```

4 Chunking based on ASP

For this subtrack, the system has to identify chunks and align them. The Inspire system realizes chunking as a preprocessing step: sentences are tokenized and processed by a joint POS-tagger and parser (Bohnet et al., 2013). Tokens, POS-tags, and dependency relations are represented as ASP facts and processed by a program that roughly encodes the following:

- chunks extend to child tokens until another chunk starts, and
- chunks start at (i) prepositions, except 'of' in 'in front of'; (ii) determiners, unless after a preposition; (iii) punctuations (where they immediately end); (iv) adverbs; (v) nodes in an appositive relation; and (vi) nodes having a subject.

These rules were created manually to obtain a result close to (Abney, 1991).

5 Experiments and Parameters

Our system does not require training, so we tested and optimized it on the training data for Headlines (H), Images (I), and Student-Answers (S) datasets. As a criteria for accuracy the competition used the F1 score based on alignments (A), alignments and alignment type (AT), alignments and alignment score (AS), and full consideration of alignment, type, and score (ATS).

Our optimization experiments showed us, that there are significant differences in annotations between datasets. In particular S contains spelling mistakes, verbs are often singleton chunks in H, and 'to' and ''s' often start a new chunk in H, while they are annotated as part of the previous chunk in I and S.

Therefore we decided to configure our system differently for each dataset, based on a Multinomial Naive Bayes Classifier trained on input unigrams and bigrams implemented using scikit-learn (Pedregosa et al., 2011). F1-score obtained on training data with 10-fold cross-validation was 0.99.

Our dataset configuration is as follows: we exclude stopwords from the calculation of similarity (1) for datasets H and I by using the NLTK corpus of stopwords; we remove non-singleton punctuation for dataset S; and we add rules to handle verb types (VBP, VBZ) as punctuation and ''s' as a preposition in chunking.

We optimized parameters for 3 runs according to different criteria.

Run 1 is optimized for the full label (ATS). We used our implementation of NeRoSim rules in the same order, except SI4, SI5, and RE1, which we excluded. In ASP this is configured by defining facts for $nextStep(s, s')$ where

$$(s, s') \in \{(noalic, equi1), (equi1, equi2), (equi2, equi3), (equi3, equi4), (equi4, equi5), (equi5, oppo), (oppo, sp1), (sp1, sp2), (sp2, sp3), (sp3, simi1), (simi1, simi2), (simi2, simi3), (simi3, result)\}.$$

Run 2 is optimized for prediction of alignment (A), this is done by using all NeRoSim rules in their original order: we define $nextStep(s, s')$ for

$$(s, s') \in \{(noalic, equi1), (equi1, equi2), (equi2, equi3), (equi3, equi4), (equi4, equi5), (equi5, oppo), (oppo, sp1), (sp1, sp2).(sp2, sp3), (sp3, simi1), (simi1, simi2), (simi2, simi3), (simi3, simi4), (simi4, simi5), (simi5, rel1), (rel1, result)\}.$$

In addition, for dataset S we perform automated spelling correction using Enchant.[3]

Run 3 is based the observation, that the scorer tool does not severely punish overlapping alignments in the F1-score of A. Hence we allow SIMI4, SIMI5, and REL1 to be applied simultaneously by defining $nextStep(s, s')$ for

$$(s, s') \in \{(noalic, equi1), (equi1, equi2),$$

[3] http://www.abisource.com/projects/enchant/

$(equi2, equi3), (equi3, equi4), (equi4, equi5),$
$(equi5, oppo), (oppo, sp1), (sp1, sp2), (sp2, sp3),$
$(sp3, simi1), (simi1, simi2), (simi2, simi3),$
$(simi3, simi4), (simi3, simi5), (simi3, rel1),$
$(simi4, result), (simi5, result), (rel1, result)\}.$

Accordingly, we expected Run 1 to perform best with respect to the ATS (and AT) metric, Run 2 to perform best with respect to A (and AS) metrics, and Run 3 to sometimes perform above other runs. These expectations were confirmed by the results shown in the next section.

6 Results and Conclusion

The results of the competition, obtained with the above parameter sets, are shown in Table 1.

The Inspire system made use of a rule-based approach using Answer Set Programming for determining chunk boundaries (based on a representation obtained from a dependency parser) and for aligning chunks and assigning alignment type and score (based on a representation obtained from POS, NER, and distributed similarity tagging). In team ranking, our system is among the top three systems for Headlines and Images datasets, and in overall ranking (both for system and gold chunks). In terms of runs (each team could submit three runs), our system obtains first and second place for Headlines with gold standard chunks. For Student-Answers dataset our system performs worst. The configuration of Run 1 performs best in all categories.

In future work we want to represent semantic knowledge in ASP externals (Eiter et al., 2015), and use ASP guesses, constraints, and optimization as outlined in (Lierler and Schüller, 2013).

Data	Run	A	AT	AS	ATS	R
Gold-Standard Chunks						
H	Base	0.85	0.55	0.76	0.55	-
	1	0.82	**0.70**	0.79	**0.70**	**1**
	2	0.89	0.67	**0.83**	0.66	**2**
	3	**0.90**	0.59	0.82	0.58	12
I	BL	0.86	0.48	0.75	0.48	-
	1	0.80	**0.61**	0.75	**0.61**	7
	2	**0.87**	0.60	**0.79**	0.59	8
	3	0.86	0.49	0.78	0.49	19
S	BL	0.82	**0.56**	0.75	**0.56**	-
	1	0.80	0.51	0.73	0.51	15
	2	0.82	0.48	0.74	0.48	16
	3	**0.87**	0.39	**0.77**	0.39	19
System Chunks						
H	BL	0.65	0.48	0.59	0.44	-
	1	0.70	**0.53**	0.66	**0.52**	4
	2	0.76	0.50	**0.69**	0.50	8
	3	**0.77**	0.46	**0.69**	0.45	13
I	BL	0.71	0.40	0.63	0.40	-
	1	0.75	**0.56**	0.70	**0.56**	4
	2	**0.82**	0.54	**0.74**	0.54	6
	3	0.81	0.45	0.73	0.45	13
S	BL	0.62	0.44	0.57	0.44	-
	1	0.69	**0.46**	0.64	**0.45**	10
	2	0.72	0.42	0.65	0.42	11
	3	**0.76**	0.34	**0.67**	0.34	12

Table 1: System Performance results (F1-score). BL is the baseline, R gives the rank in the respective competition category. Abbreviations are explained in Section 5.

Acknowledgments

This research has been supported by the Scientific and Technological Research Council of Turkey (TUBITAK) Grant 114E777.

References

Steven P Abney. 1991. *Parsing by chunks*. Springer.

Eneko Agirre, Aitor Gonzalez-Agirre, Iñigo Lopez-Gazpio, Montse Maritxalar, German Rigau, and Larraitz Uria. 2016. SemEval-2016 Task 2: Interpretable semantic textual similarity. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval 2016)*, San Diego, California, June.

Rajendra Banjade, Nobal B Niraula, Nabin Maharjan, Vasile Rus, Dan Stefanescu, Mihai Lintean, and Dipesh Gautam. 2015. NeRoSim: A system for measuring and interpreting semantic textual similarity. In *SemEval 2015*, pages 164–171.

Steven Bird. 2006. NLTK: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics.

Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richárd Farkas, Filip Ginter, and Jan Hajič. 2013.

Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428.

Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. 2011. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103.

Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. 2012. ASP-Core-2 Input language format. Technical report.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2014. One billion word benchmark for measuring progress in statistical language modeling. Technical report.

Thomas Eiter, Michael Fink, Giovambattista Ianni, Thomas Krennwallner, Christoph Redl, and Peter Schüller. 2015. A model building framework for Answer Set Programming with external computations. *Theory and Practice of Logic Programming*, FirstView:1–47.

Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. 2011. Potassco: The Potsdam Answer Set Solving Collection. *AI Communications*, 24(2):107–124.

Yuliya Lierler and Peter Schüller. 2013. Towards a tight integration of syntactic parsing with semantic disambiguation by means of declarative programming. In *International Conference on Computational Semantics (IWCS)*, pages 383–389.

Vladimir Lifschitz. 2008. What is answer set programming?. In *AAAI*, volume 8, pages 1594–1597.

Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

George A Miller. 1995. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830.