FINAL REPORT

VerbMet: Verbal Metaphor Annotation and Identification

Molly Moran, Samantha Richards, and Arjuna Mahenthiran

Code and Documentation available at:
https://github.com/mgmoran/VerbMet

Table of Contents

I. Background & Motivation

Automatic detection of figurative language is not a new problem in NLP, but it is far from solved. Figurative language is pervasive across all styles and genres of text - so much so that human beings sometimes have difficulty recognizing it - so learning to recognize and parse it properly is essential to building an effective language model.

Metaphors are one subtype of figurative language that involve an implicit or explicit mapping from one semantic domain (a "source" domain) to another (the "target" domain.) Much of the modern literature about automatic metaphor detection ascribes to the metaphor taxonomy outlined by Krishnakumaran and Zhu[1], in which there are three distinct types of metaphor: type I compares a noun to another noun using a copula, type II uses an adjective to modify a noun metaphorically, and type III involves a metaphorical relationship between a verb and a noun.

We found the least available literature on type III. Our intuition is that verbal metaphor can be identified more specifically as a usage of a verb where there exists some sort of semantic "mismatch" between a verb and (at least) one of its arguments. The locus of this mismatch varies, and we believe that it varies systematically depending upon the verb. That is, verbs tend to exhibit selectional preferences for where a semantic mismatch is likely to occur if they are used metaphorically. When an intransitive verb is used metaphorically, we *almost* always see a subject-verb mismatch:

Evening fell.

But we may also see mismatches between verb and adjuncts:

We were swimming in cash.

For transitive verbs, there are more possibilities. There might be a mismatch of subject and verb:

The wind whistled.

Or there might be a mismatch of verb and object:

[1] Krishnakumaran S, Zhu X (2007) Hunting elusive metaphors using lexical resources. In: Proceedings of the Workshop on Computational Approaches to Figurative Language (FigLanguages '07). Stroudsburg, PA: Association for Computational Linguistics. 13–20.

'Tis the season to spread cheer.

Or it might be one of any possible adjuncts:

She carried me to safety.

If we can obtain reliable data about which kind of mismatch a particular verb selects for, we may be able to reduce the amount of work a machine-learning algorithm needs to do in order to detect a figurative usage of that verb.

II. Corpus

Because we hoped to investigate a verb's relationship to its arguments as an indicator of literal or nonliteral usage, and extracting a verb's arguments is a nontrivial task, we chose to use a corpus already tagged with verbal arguments. This gave us the advantage of being able to design a more powerful classifier--removing the uncertainty relating to detecting the argument--albeit with the not-insignificant caveat that our classifier would be restricted to uses on unseen data already pre-annotated (in the same manner) for verbal argument. However, we considered this an acceptable tradeoff given that, between the handful of leading semantic annotation styles, increasing amounts volumes of linguistic data may be annotated in this way in the future. We therefore chose to work with the Wall Street Journal corpus, consisting of ~30,000,000 words spanning three years of Wall Street Journal text from 1987-1989. This corpus is already configured with syntactic tagging via Penn Treebank and semantic tagging via PropBank, which includes main verb and argument identification.

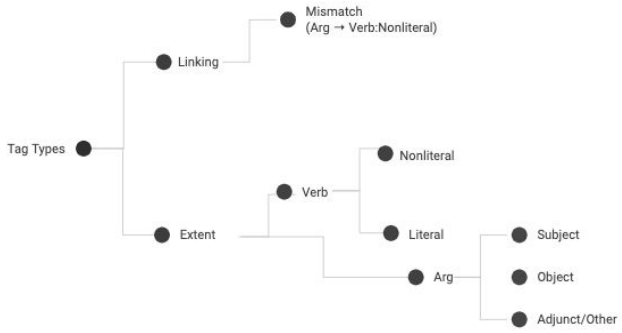
Our annotation task was designed to be done at the sentence level, not just the verb-phrase level, such that all arguments of a given verb could be present for analysis. Individual sentences for annotation were extracted from PropBank "instances", objects that bundle all of the pre-existing semantic and syntactic annotation for a given WSJ sentence.

In an attempt to avoid confusion about what constitutes a main verb, and to help ensure that our annotators picked the same verb to annotate as often as possible, the we filtered the PropBank instances for those corresponding to sentences that contained only a *single* main verb. This filtering gave us ~900 total sentences. Each annotator was given 300 total sentences to annotate. These batches each consisted of 200 unique sentences, and a final 100 sentences that were shared among all three annotators.

III. Annotation

i. Schema

Our annotation schema consists of two separate but related tasks. The first task is a binary classification for each main verb - is its use in context Literal or Nonliteral? The second task applies to Nonliteral verbs only, and involves identifying semantically 'mismatched' arguments contributing to the nonliteral reading of the verb in context. Mismatched arguments are tagged with their type - "Subject", "Object" and "Adjunct/Other", and a "mismatch" linking tag is added from a given argument to a main verb. The below diagram represents our schema:



Annotators were instructed to tag main verbs *only.* This excluded copula, auxiliaries, and in the case of phrasal verbs, anything other than the main verb. For arguments, annotators were instructed to tag the *entire span* of an argument NP rather than just the head. This decision was made to maximize the possibility of recovering the correct argument if there were inconsistencies in tagging.

Finally, we had an adjudicator annotate the 100 sentences of "shared" data. We chose to use the shared data of the annotator who agreed most highly with our Gold Standard annotations.

ii. Revisions

Our original schema looked very similar to this final form with one key exception: the "mismatch" linking tag was added during the revision process. In some ways, the mismatch tag might appear redundant: after all, under the assumption that each sentence contained only a single main verb, if an argument is annotated at all in a given sentence, it is implied that the argument mismatches the (one and only) main verb of that sentence. We ended up adding the "mismatch" tag as a failsafe, to ensure that we would be able to successfully recover the specific verb-argument relationship the annotator wanted to convey, in case of multiple verbs being tagged.

iii. Inter-annotator agreement

In order to accurately calculate agreement, we had to do some pre-processing on the shared annotations. First, we had to remove all copula and auxiliaries that were accidentally tagged during the course of annotation. Additionally, sometimes either PropBank or annotators mis-identified participles or gerunds as "main verbs", leading to occasional disagreement between annotators about the identity of the main verb in the sentence. For consistency, we kept only those annotations wherein all three annotators tagged the same main verb.

We calculated agreement on two levels, corresponding to our two annotation tasks: first, a binary value of whether or not the annotators assigned the same label ("Literal" or "NonLiteral") to a given verb. Second, we used MASI-distance to calculate the distance between the sets of arguments tagged by each annotator as "mismatched" for a given verb. We calculated the following agreement measures:

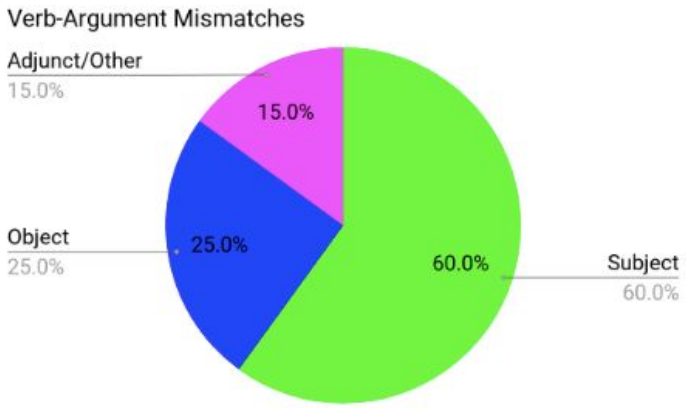
Observed Agreement	S	Pi	Fleiss' Kappa
.89	.88	.78	.78

Again, for consistency, when calculating MASI-distance between sets of tagged arguments, we made sure to only calculate this distance with respect to verbs for which all three annotators marked the verb as nonliteral. This decision was made in order to avoid getting wires crossed between our two planes of agreement.

iv. Stats

The diagrams below summarize the results of our annotated data:

"Literal" tags	"Nonliteral" tags	Unique Verbs	Verbs with both Nonliteral and Literal instances
753	239	199	46



The mismatched-argument statistics were the most crucial to our classification approach. Using these data, we created a dictionary mapping every nonliteral instance of a given verb token to the type of argument identified as the semantic mismatch in that particular context. For example, if we encountered the verb 'rise' as nonliteral 5 times, and 3 of those times involve a mismatched subject, and the other 2 involved a mismatched object, the dictionary entry for 'rise' might look something like this:

[Subject,Object,Object,Subject,Subject]

We then took the argument type with the maximum count to obtain the "most likely mismatched argument", data which would later be used to train one of our classifiers.

IV. Data Pre-Processing

i. General pre-processing

First, from our annotated data, we created a dictionary of dictionaries. In the outermost dictionary, the document name within PropBank was the key, and the values were dictionaries. For each of these inner dictionaries, the key was the sentence number within the document and the value was the text of that sentence. Using this, we were able to create another dictionary in which the text of each sentence within our data served as the key and its PropBank instance served as the value.

Having found the PropBank instance for each sentence within our data, the next step was accessing information about the verb and its arguments. This meant having to search the subtrees for both the predicate and the arguments as mapped by PropBank. Unfortunately, PropBank does not mark the heads of constituents within their trees, so a rough method of finding the head noun was used; that is, using the rightmost NP or NN within the phrase.

ii. VerbMetObjects

One of our main goals in selecting PropBank was to be able to take advantage of its argument and predicate information, but in order to do this we needed to extract this semantic/syntactic information from PropBank's objects. As noted above, each sentence in the WSJ database is mapped to a PropBank 'instance', and some of our initial preprocessing (discussed in the above section) was devoted to mapping these instance objects to the labeled data generated from our annotations in MAE. Once we had obtained these instances, the pre-identified (PropBank-annotated) arguments and predicates for each sentence are available as Treepointer objects, which can ultimately be used to obtain a subtree representation of the argument.

Since we were dealing with NPs as arguments, this meant that we ended up with some complex syntactic constituents that needed to be parsed to obtain a string representation of the head N. Parsing each NP to obtain the head N, which we did using a small rule-based algorithm, gave us the argument strings, which we would be able to feed into the classifier. However, given the relatively small amount of data that we were working with, we also hoped to be able to obtain from PropBank (in addition to reliable identification of the arguments) more abstract semantic information about the senses of the verbs in relation to their arguments for each particular instance of use. Luckily, PropBank instances are also tagged with a class of objects called 'rolesets', groupings that provide rough word sense information for the predicate, including providing sense-specific 'roles' for each argument. For example, the roleset for the instance of 'join' in our favorite WSJ sentence contains the following argument information (PB-identified arguments in green):

> Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.
>
> join.01
> 0 agent, entity doing the tying
> 1 patient, thing(s) being tied
> 2 instrument, string

It is worth noting that because these are abstractions, the rolesets don't give us information about the meaning of the arguments in this specific instance; however, they provide general information about what arguments might be expected to do in this or similar senses of the predicate *join.* After extracting rolset information, we wrapped it and the argument information noted above into a VerbMetObject. VerbMetObject class was essentially a convenient way to bundle together and render easily and efficiently accessible all of the information that we hoped to feed into our naive bayes classifier as features.

iii. Google vectors

Intuitively, it seems to be the case that mismatches resulting in metaphors are likely to occur where there are also mismatches between the domain of the verb and its argument. For example, in "The *wind* whispered a secret", the mismatch in the domain of animacy between "wind" and "whispered" causes the verb to be used metaphorically. However, a problem that presents itself is trying to determine whether two words are within the same domain or not. Unfortunately, there does not seem to be a robust enough data-set with strictly defined domains as well as categorizations for enough nouns to be helpful. As a potential remedy to this, we opted to use vector semantics.

Vector semantics provides a useful way for describing how similar two words are based on the contexts in which they most frequently appear. While this is not a perfect method of comparing word similarity, it is the case that words with more similar connotations are more likely to appear in similar contexts, e.g. "king" and "queen" as opposed to "king" and "cashier". It is using this method that we hope to incorporate some of the semantic information about the words involved in creating metaphors in order to better construct a classifier for metaphors.

In order to properly use vector semantics, it is first important to have a vector for each word that is robust; that is, it has many values filled for a variety of contexts, based on a large amount of data. To this end, we struck upon using Google vectors. These vectors were constructed by Google and are based upon the contexts within which words are found from a variety of sources. The specific text file used of Google vectors was received from an assignment for CS114, and is almost 1GB worth of data. While this intimates just how rich Google vectors are, it is also important to note that this is the reason why the specific text file we are reading in cannot be found on Git along with the rest of our project: it is simply too big.

Using Google vectors, we first had to find the vector associated with the main verb in the sentence. We considered using a lemmatizer on the verbs and arguments within each sentence in order to increase the likelihood of them being found within Google vectors, however ultimately decided against it as it might have created other issues or loss of information that might have been more detrimental to our data. Therefore, in instances where the main verb in the sentence was not present in the Google vectors, the sentence was simply ignored and not included within the training or testing data. After finding the vector associated with the verb, the cosine similarity to each of the arguments was computed using their Google vectors as well. At this step, for instances where the argument could not be found in PropBank or if the argument did not exist in the sentence (e.g. a direct object for an intransitive verb), -1 was simply considered the value. All of these values were then themselves stored in a vector for each sentence, and along with its annotator-labeled classification, used to train the model.

V. Machine Learning Baseline: K-Nearest Neighbors

i. Motivation

The majority of previous literature on this subject uses involves classifying metaphor using word embeddings and neural nets. However, most of these attempts involved detecting adjective-noun and noun-noun comparisons. As mentioned previously, the lack of literature on detecting verbal metaphor is not surprising given that this type of metaphor appears to be more semantically complex than either of the first two. K-Nearest Neighbors provides a powerful machine-learning mechanism for making predictions and classifications: by plotting values from training data within Euclidean space and bearing in mind their classification, K-Nearest Neighbors is essentially able to fit a model in which there are "zones" of each classification corresponding to each label. However, considering that K-Nearest Neighbors needs numerical input in the form of a collection for each data-point, it seemed a prime choice for our forays into vector semantics. Finally, for each element in the testing data, K-Nearest Neighbors plots the point in the same space as the model and takes a poll of those K (some number) points from the training data closest to the testing datum. From this poll, it is able to determine then what label to assign to the test instance.

ii. Results

The confusion matrix belows shows the results from a trial of the K-Nearest Neighbors classifier. From this, it is clear that it is much better at classifying 'Literal' verb instances than 'Nonliteral' verb instances. This reaffirms the notion that while vector semantics may prove useful when comparing elements of similar syntactic categories, it is hard to abstract beyond category boundaries (i.e. verb to noun). Similarly, this also indicates that there may have been a much wider range in values of cosine similarity between main verbs and arguments in nonliteral cases than in literal cases, making it harder to classify these properly. The overall accuracy indicates that it performed only slightly better than random chance, which is probably due to its efficacy in solely identifying literal cases.

Confusion Matrix

	Precision	**Recall**	**F1**	**Accuracy**
Literal	**.76**	**.73**	**.75**	**.62**
Non-Literal	**.22**	**.24**	**.23**	

VI. Machine Learning II: Naive Bayes

We chose Naive Bayes to work with as our second model because of its relative ease of implementation, and its reliability for smaller datasets (of which

ours certainly was). It is quite simple to add and quickly evaluate a variety of features. We used nltk's naive bayes classifier.

i. Features

After some tuning of our features based on dev set evaluation, the final feature set consisted of a combination of general features and mismatch-dependent features. The argument-specific features were either abstract semantic argument description strings taken from the roleset for the sentence predicate or boolean variables indicating whether or not the verb had an arg0, arg1, or arg2 in the context of the data. Although we had initially incorporated as features context-specific information--namely, the verb and argument strings--we actually found these features to be either (a) irrelevant for our performance (in the case of the verb string) or (b) detract from our performance (in the case of the argument strings). This was perhaps not surprising given the small size of our dataset, where specific instances of verbs and arguments are unlikely to be as informative as more abstract representations of their semantic characteristics.

At first, we tried incorporating the rolesets of all arguments as features for every verb. This yielded a 70% accuracy. However, we were interested in what the information on the locus of mismatch could tell us, and so decided to restrict these features to only the the most likely mismatched argument. That is, the argument-specific feature was added only if our annotators indicated that the mismatch was with that argument--the idea being that this would provide some focused guidance for the model as it learned from the argument-related features we gave it. This resulted in an accuracy of 78%, which tells us that our intuition was correct, and narrowing the focus to a particular argument does not result in any loss of information: in fact, it improves our accuracy.

General features used were the verb string itself, whether the subject (arg0) of the verb was a proper noun, and whether the verb had an arg2 argument (also the result of some exploratory investigation of the most useful features).

ii. Results / Most informative features

The confusion matrix displaying the results of the model is shown below. The clear pattern is that the classifier's performance on the literal is significantly better than its performance on the nonliteral, with F1 at .88 for Literal and only .66 for Nonliteral This is probably because there were more instances of literal verbs. However, on both categories the model performed significantly above chance, which was impressive given the small amount of data, limited training of annotators, and difficulty of the task.

Confusion Matrix

	Precision	**Recall**	**F1**
Literal	**0.84**	**0.93**	**.88**
NonLiteral	**0.77**	**0.59**	**.66**

Most informative features

Feature	Predictive of	Informativeness
Has an arg1	Nonliteral	14:1
Generally arg0 is a 'sayer'	Literal	9:1
Has an arg0	Nonliteral	8:1
Generally arg1 is a 'thing'	Nonliteral	8:1
General arg1 is 'Logical'	Nonliteral	6:1

'Generally' refers to the fact that the feature derived its argument information from the roleset, not the specific argument in the context of the data. One drawback is evident here in the identity of the second most informative feature: one of the most common verbs in the dataset was 'say', and say was always literal in usage. The reason this feature is useful, then, is solely because of the high frequency of 'say' and its exclusive occurrence as a literal verb, making classifying this verb quite an easy task for the model. Ideally we would have liked to control for such cases of frequent verbs with single-label occurrence in the data.

VII. Concluding Thoughts

V-N metaphors are very different from N-N and A-N metaphors and thus we need different approaches for classifying them. First and foremost, having a reliable method to identify the verbal argument is critical, although it does come at a cost. As discussed above, our decision to use a pre-annotated dataset significantly restricts the use of our classifier to datasets that have been annotated with PropBank predicate and argument tags. Yet our findings from our evaluation of the Naive Bayes Classifier support the notion that the secondary advantage of using a PropBank-annotated dataset, that of having access to more abstract semantic information about the relation between predicates and arguments, in giving us the capacity to abstract away from context-specific information was

actually quite valuable, and perhaps a way to cope with a relatively small amount of data. Although the performance of the nearest neighbors classifier, incorporating vector distance of each verb from its argument, was relatively poor compared to naive bayes, it still performed above chance, and if anything, represented the ultimate test of the utility of abstract semantic information. In future work it would be interesting to fold a vector semantics feature into a Naive Bayes featureset to see if it may provide the model with additional untapped information.

Ultimately, the project was most challenged by two factors: (a) the limited resources with respect to annotator time and dataset size and (b) the relatively restricted range of information that we were able to put towards features for our classifier. A general issue faced when performing machine learning based on human annotation is that of a paucity of data; of course in an ideal world, annotators would be able to tag vast quantities of data in order to ensure precise results. However, given the realistic constraints on resources, whether that be time such as in our procedure or perhaps money in another, it is difficult to obtain enough data to satisfy every inquisitive itch. Particularly in our methodology, this played a factor in creating our training and test sets: because we only had ~600 annotated sentences in the end to use, creating a test set of 10% would result in a size of around 60 sentences. The problem is that in a set this small, a chance number of more correct classifications can result in an artificially inflated accuracy, and the opposite is true as well. To remedy this here, we created test sets that were larger, at 33% and 30% for K-Nearest Neighbors and Naive Bayes, respectively. While this did help remedy the issue of random chance having a large impact on the data, we were left wondering if some of our more unsatisfying results might be the fault of overfitting the data. Therefore, in future research and generally a more ideal world, we would still be able to use a small test set relative to the training set without being too affected by chance.

Secondly, metaphor detection is a particularly challenging problem because unlike tasks like sentiment analysis, the spans of text from which useful features may be drawn is quite limited. Since verbal metaphor specifically derives from a mismatch between the verb and its arguments, features must be obtained from some combination of the verb and its arguments, whether from the context-specific occurrences, semantic abstractions like rolesets, frame files, or levin classes, or vector semantic representations. Since the task itself, annotation of verbal metaphor between a verb and its nominal arguments, defines the part of speech, even syntactic information cannot be used. This requires some amount of creativity and resourcefulness on the part of the computational linguistic when designing the features of the machine learning model.

Despite these drawbacks and challenges, the fact that we were able to use even a limited set of tools--Propbank annotations and Google vector semantics--to generate above-chance predictions illuminates a hopeful path forward for models that can detect verbal metaphor. We have learned, for instance, that encoding

information relating to most likely mismatch can guide the model to select more useful argument features. With more data, more resources, and more hard work, the daunting figure of metaphorical language in AI may diminish to an increasingly more manageable size.