Structured representations

The dative alternation

In this assignment, we'll perform a simple study of the dative alternation. We already looked at important factors that help predict the dative alternation in the \( R \) section of the course, but that data was already processed into a CSV file. If you wanted to do a corpus study of your own, building that file would be your first step.

We'll look at the kind of syntactic analysis you'd need to conduct such a study and see how to use the \textit{NLTK} library to implement it. \textit{NLTK} offers custom libraries for working with a variety of formats you might encounter in popular corpora (including XML and the Praat TextGrid format).

Simple formats

So far, we've gotten a lot done with datasets that are fairly simply formatted. \textit{Alice in Wonderland} was plain English text. Everything in the \textit{Alice} file was part of the actual text of \textit{Alice in Wonderland}.

The \textit{Fisher} dataset had a little more formatting. Fisher had lines like this:

\begin{verbatim}
56.38 61.03 A-f: i mean no money is very important definitely and a million dollars is a dream come true for me i mean
61.19 64.71 A-f: i can actually imagine the kind of things i can buy
64.41 65.38 B-f: yeah
65.57 69.14 A-f: travel and do whatever i want to in life but then again
69.61 70.13 A-f: i just
70.25 73.34 A-f: can't i mean stop talking to my best friend that's like
\end{verbatim}

Not all the elements in these lines are part of the actual utterances, but you can tell \textit{data} (the utterances) from \textit{metadata} (information about the utterances) by splitting things up based on spaces and looking at positions. The dative alternation dataset you dealt with in \( R \) worked the same way.

The Penn Treebank

Not all datasets work well with this kind of simple format. For instance, what if you wanted to do a corpus study of the dative alternation? You could just search for patterns like “give him a”, “sell her the”, etc. but this approach has some disadvantages. (What are they?)

It would be nicer to work with \textit{syntax trees} which represent the structure you're looking for directly. But trees aren't quite as simple to write down.

We'll start off by looking at the most influential corpus of parsed English data, the \textit{Penn Treebank}. The Penn Treebank consists of about a million words of Wall Street Journal news articles with parse trees hand-corrected by UPenn graduate students to make sure they're correct. It's a pretty good resource for studies of common constructions like the dative alternation.

The Treebank uses a version of Principles-and-Parameters grammar that might strike some of you as old-fashioned. But it's what we've got--- if you preferred to do your corpus study using some other grammatical theory, you'd have to pay more graduate students to annotate more data, or try to automatically translate the Treebank into your favorite notation. (People have done this for CCG and HPSG, with some success.) For the most part, for corpus studies of constructions like dative alternation, the different theories of syntax won't matter all that much.

Let's start out by downloading the Penn Treebank data and taking a look at it from the command line. Using \textit{ls}, it's straightforward to find that there are 24 directories, each one containing about 100 \textit{.mrg} files.

What's in the files? Well, let's take a look:
Despite the fancy .mrg suffix, the files contain text data (not some kind of tricky binary encoding). But the data is in a special format which gives the structure of the syntax tree. Each constituent in the tree is contained within parentheses, in the format (LABEL child1 child2 child3...).

For instance, let's look at the noun phrase "61 years". In Penn Treebank notation, a noun phrase is labeled NP, so we begin with a parenthesis, then the label NP. After that, we have the two children, "61", which is labeled CD (cardinal number) and "years", which is labeled NNS (plural noun). And finally we have the closing parenthesis.

"61 years" is itself part of the phrase "61 years old", which is an ADJP (adjectival phrase) that also contains the JJ (adjective) "old".

Outside the tree is a final bracket with no label which contains the entire sentence.

Look at another tree in the data files and draw out a medium-sized subtree on paper. Does the Treebank parse things the way you'd expect?

**Reading the Treebank**

In the next part of the unit, we're going to use a library to read in Treebank data and search it for datives. But before we use the off-the-shelf solution, take a minute to think about how you'd read in Treebank trees "by hand". How would you represent a tree as a Python data structure? What functions would you write to read the text format and transform it into your data structure? Think it over before you go on.

Here's a simple proposal: represent the tree using Python lists, with a list for each constituent. For instance, the adjective phrase would be:

```
["ADJP", ["NP", ["CD", "61"], ["NNS", "years"]], ["JJ", "old"]]
```

What are the advantages of this kind of structure? What are the disadvantages? How would you build it?

**A Treebank reader**

Although we won't spend any more time on the hand-built solution, you may want to glance over a function that constructs it:

```python
def readTree(text, ind, verbose=False):
    """The basic idea here is to represent the file contents as a long string and iterate through it character-by-character (the 'ind' variable..."
```
points to the current character). Whenever we get to a new tree, we call the function again (recursively) to read it in."

if verbose:
    print("Reading new subtree", text[ind][:10])

# consume any spaces before the tree
while text[ind].isspace():
    ind += 1

if text[ind] == "(":
    if verbose:
        print("Found open paren")
    tree = []
    ind += 1

    # record the label after the paren
    label = ""
    while not text[ind].isspace() and text != ")":
        label += text[ind]
        ind += 1

    tree.append(label)
    if verbose:
        print("Read in label:", label)

    # read in all subtrees until right paren
    subtree = True
    while subtree:
        # if this call finds only the right paren it'll return False
        subtree, ind = readTree(text, ind, verbose=verbose)
        if subtree:
            tree.append(subtree)

        # consume the right paren itself
        ind += 1
        assert(text[ind] == ")")
        ind += 1

    if verbose:
        print("End of tree", tree)

    return tree, ind

elif text[ind] == ")":
    # there is no subtree here; this is the end paren of the parent tree
    # which we should not consume
    ind -= 1
    return False, ind

else:
    # the subtree is just a terminal (a word)
    word = ""
    while not text[ind].isspace() and text[ind] != ")":
        word += text[ind]
        ind += 1

    return word, ind
if verbose:
    print("Read in word:", word)

return word, ind

if __name__ == "__main__":
    ff = open("/home/corpora/original/english/penn_treebank_3/parsed/mrg/wsj/00/wsj_0004.mrg")
    filetxt = "".join(ff.readlines())

    #read all the trees in a file
    ind = 0
    while ind < len(filetxt) - 1:
        tree, ind = readTree(filetxt, ind)
        print(tree)
        # print "new ind", ind

There are a few things you can learn from this program. One is that a sizable chunk of code like this can be fairly difficult to understand! (Running it in verbose mode might make things a little easier to figure out. Can you see how to do that?)

Another is the idea of recursion--- functions that call themselves. Recursion is a powerful mechanism for computing complicated things. It allows the function to build up a tree made of subtrees, which are in turn made of subtrees, and so on. (Of course, one has to avoid the "turtles all the way down" style of recursion which never terminates; in this case, we stop when we reach the words.) Using recursion properly is tricky, so we won't look any more closely at it for now.

Reading the Treebank with library functions

How would we find a library that deals with the Treebank data for us? Well, try googling "penn treebank python"! You should get a bunch of pointers to the NLTK library, which is a large suite of tools for natural language processing in Python.

Can you find out the names of the modules you need to use? What are some key functions?

One way to do read trees is with nltk.corpus.BracketParseCorpusReader. Take a look at the help text for this module. Like str and list, BracketParseCorpusReader is an object. An object is a programming abstraction for some data and associated methods. You can think of it as a kind of machine or device stored in computer memory. To use an object, you construct it (make a device with the particular settings you need) and then call its methods (use the device to do things).

For instance, you can make a string by writing it with quotes, or by using the str function on an object of some other type:

```python
>>> str(10)
'10'
```

And you can do things with the string you've just made, like split it, or replace part of it:

```python
>>> "hello, world".replace("hello", "goodbye")
'goodbye, world'
```

Let's look at the help text again:

```python
class BracketParseCorpusReader(nltk.corpus.reader.api.SyntaxCorpusReader)
    Reader for corpora that consist of parenthesis-delineated parse trees.

    Method resolution order:
    BracketParseCorpusReader
```
The __init__ function constructs a BracketParseCorpusReader; don't worry about the first argument (self), which is a placeholder for the reader you're trying to make. The documents do reveal that a BracketParseCorpusReader takes the name of a directory and a list of files, or a regular expression that matches these files. (Remember regular expressions from grep? This is one of the rare cases in which using one doesn't create more problems than it solves.)

Let's try this out:

```python
import nltk.corpus

def parsedSents(wsjDir):
    reader = nltk.corpus.BracketParseCorpusReader(wsjDir, ".*/.*\.mrg")
    return reader.parsed_sents()

if __name__ == "__main__":
    wsj = "/home/corpora/original/english/penn_treebank_3/parsed/mrg/\n    trees = parsedSents(wsj)
    for ti in trees:
        print(ti)
```

The critical line is: reader = nltk.corpus.BracketParseCorpusReader(wsjDir, ".*/.*\.mrg") which constructs the reader object. The first argument is the directory path and the second is a regular expression. What does it match? Why is this useful?

Once we've constructed the reader, we use its parsed_sents method to return all the sentences in the corpus. As you've already seen, object methods use the object.method(args) syntax; BracketParseCorpusReader is no exception.

What does this program print when you run it? What is the type of the objects in the trees structure? How can you find out more about this type?

**Trees!**

To figure out a little bit more about the Tree class, let's look at a smaller tree close up. Start out by creating the following data file

```
( (S (NP (DT the) (NN cat))
  (VP (VBD sat)
    (PP (IN on)
      (NP (DT the) (NN mat)))
  )
)
```

We'll use the following lines to read it in:
Reading the documentation, we can see that a Tree has a bunch of different methods that do various things. Among them, we can find the label at a node of the tree, the children below that node, and the list of words at the terminals. How do we do these things?

A Tree is made up of more Tree--- for instance, the first child of the first child is tree[0][0]. What is this? (You can also write Tree[0,0]; this is a special feature of the Tree class and doesn't work on regular lists.)

Interlude: syntax

We could immediately launch ourselves into playing with the Tree class and exploring all its many capabilities. But before we immerse ourselves in low-level details, let's switch back to linguist mode and think about what we want our program to eventually do. We'd like to find sentences which contain double object ("give him a ball") and prepositional ("give a ball to her") dative constructions.

Suppose we could build some kind of "dative detector" function--- actually two functions, one for double object and one for prepositional datives. We'd run these on each sentence in the treebank and print out the matches for further analysis. For instance, we could print out factors like the identity of the verb, or whether the recipient was a pronoun, and then do statistical analysis of the relationships between them.

What would this dative detector have to do? Think of some examples of double object datives and prepositional datives yourself. What do you think their trees would look like?

To develop a program of this type, I'd normally put together a scratch file containing a few possible targets and non-targets and use this for initial testing. To do so, I'd search the corpus files for obvious cases using less or a word processor. (How do you search for things in less? Type the forward slash / and then a string--- actually, a regular expression--- to search for.)

What should we search for to maximize our chances of finding datives?

Here are a couple of sentences I found (and simplified) from the corpus:
(PP (IN for)
  (S-NOM
    (NP-SBJ (-NONE- *) )
    (VP (VBG maintaining) (CC or) (VBG increasing)
      (NP (NN ad) (NN spending) )))
  (VP (VP have)
    (NP-PRED
      (NP (JJ permanent) (NNS fixtures) )
      (PP-LOC (IN at)
        (NP (DT the) (NN news) (NNS weeklies) )))
  )
  (VP (VBP have)
    (VP (VBN become)
      (NP (JJ permanent) (NNS fixtures) )
      (PP-LOC (IN at)
        (NP (DT the) (NN news) (NNS weeklies) )))
  ))
(S (NP-SBJ-3 (NNS teachers) )
  (VP
    (ADVP-TMP (RB sometimes) )
    (VB give)
    (PRT (RP away) )
    (NP (DT a) (JJ few) (JJ exact) (NNS questions)
      (CC and)
      (NNS answers) )
  )
  (. .))
(S (NP-SBJ-1 (DT The) (NN show) )
  (VP (VBD did) (RB n't)
    (VP (VB give)
      (NP
        (NP (DT the) (NNS particulars) )
        (IN of)
        (NP
          (NP (DT the) (NNS particulars) )
          (IN of)
          (NP
            (NP (NNP Mrs.) (NNP Yeargin) (POS 's) )
            (NN offense) ))
        )
      )
    )
  )
  (. .))
(SINV
  (S-TPC-2
    (NP-SBJ-1 (DT The) (NN deal) )
    (VP (VBZ is)
      (ADVP (RB chiefly) )
      (VP (VBN designed)
        (NP-3 (-NONE- *-1) )
        (S-CLR
          (NP-SBJ (-NONE- *-3) )
          (VP (TO to)
            (VP (VB give)
              (NP (NNP Mitsubishi) )
              (NP
                (NP (DT a) (NN window) )
                (PP-LOC (IN on)
                  (NP (DT the) (NNP U.S.) (NN glass) (NN industry) ))))))))
  )
  (. ,)
  (VP (VBZ says)
    (S (-NONE- *T*-2 )
      (NP-SBJ
Which ones are double object datives? Which ones are the prepositional datives? Which ones aren’t datives at all?

Let’s put these in a file (call it “testfile.mrg”) and then build a program that can find the constructions we’re looking for.

### Detecting datives

Ok, let’s try this out. We’ll start off with NP-PP datives. From your analysis of the corpus, you’ve probably figured out that these have a VP containing an NP followed by a PP-DTV. Let’s build a function, ppDative, that detects this configuration:

```python
def ppDative(tree):
    #return True if tree contains a prepositional dative
    #(code goes here)
    return True
```

If you have some idea of how to start out building this, go ahead and do it on your own. If not, keep reading.

My target configuration starts off with a VP, so I want to check all the subtrees and see which ones are VPs. I hunt through the NLTK documentation and find the `subtrees` function--- this looks useful!

```python
| subtrees(self, filter=None) |
| Generate all the subtrees of this tree, optionally restricted |
| to trees matching the filter function. |

>>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP (D the) (N cat))))")
>>> for s in t.subtrees(lambda t: t.height() == 2):
...     print s
...```

Using this, write some code that finds all the VP nodes and prints them out. (There are two ways to do this. I did it the syntactically simpler way, but you can use the filter function argument if you like.) The answer is below if you can't figure it out.

```python
def ppDative(tree):
    for st in tree.subtrees():
        if st.label() == "VP":
            print("matched vp", st)
```

Now we want to find out if the node has an NP child followed by a PP-DTV child. To find out, we can use a for loop over the children:

```python
for ch in range(len(st) - 1):
    if (st[ch].label().startswith("NP") and
        st[ch + 1].label() == "PP-DTV"):
        print("matched dative", st)
```

Look at the output. Does it do what you expect? Add a return True statement and see whether you get the right cases.

My program so far looks like this:

```python
import nltk.corpus

def parsedSents():
    reader = nltk.corpus.BracketParseCorpusReader(".", "testfile.mrg")
    return reader.parsed_sents()

def ppDative(tree):
    for st in tree.subtrees():
        if st.label() == "VP":
            #print("matched vp", st)

            for ch in range(len(st) - 1):
                if (st[ch].label().startswith("NP") and
                    st[ch + 1].label() == "PP-DTV"):
                    #print("matched dative", st)
            return True

trees = parsedSents()
for t in trees:
    print(" ".join(t.leaves()))
    if ppDative(t):
        print("\tPrepositional dative")
```

And prints this:
Looks ok. Let's try the double object dative. My target pattern here is NP followed by NP under VP. I come up with the following:

```python
def npDative(tree):
    for st in tree.subtrees():
        if st.label() == "VP":
            #print("matched vp", st)
            for ch in range(len(st) - 1):
                if (st[ch].label().startswith("NP") and
                    st[ch + 1].label() == "NP"):
                    # print("matched dative", st)
                    return True
```

What are the results like on our sample file?

**Iterative debugging**

Since it seems to work out okay, let's try it out on the treebank. This will spew a lot of output very quickly. To scroll through it line by line, we can pipe it to `less`. Remember that the pipe `|` takes the output of one program and redirects it to another. (And if you forget what `less` does, you can check the manual!)

```
$ python3 dativeDetector.py | less
```

Scroll through the output and look for prepositional datives:

```
Then, just as an image of the statue of Thomas Jefferson dissolves from the screen, the announcer continues:
"On the issue of abortion, Marshall Coleman wants "-1 to take away your right " to choose and "-1 give it to the politicians."
Prepositional dative
While "-1 giving the Comprehensive Test of Basic Skills to ninth graders at Greenville High School last March 16,
she spotted a student looking at crib sheets.
Prepositional dative
Worksheets in a test-practice kit called "Learning Materials", sold " to schools across the country by
Prepositional dative
"It "as if France decided "-1 to give only French history questions to students in a European history class,
and when everybody aces the test "T"-3, they say O their kids are good in European history," says "T"-2 John Cannell,
an Albuquerque, N.M., psychiatrist and founder of an educational research organization, Friends for Education,
which "T"-105 has studied standardized testing.
Prepositional dative
```

Three of these hits look good:

- "take your right to choose and give it to politicians",
- "giving the Test of Basic Skills to ninth graders",
- and "give only French history questions to students".

But one of them isn't so good: "Worksheet sold to schools" is a passive construction. Does this construction have a corresponding double object dative?

What about the double object dative? Does this also have a problem with passives?

Here's a corresponding case:
At Cray Computer, he will be paid $240,000.

We'll need to detect passive sentences so we can filter them out (or, if we prefer, to report them but mark them with a special feature so we don't analyze them as if they were active).

So let's look at some trees for our problem cases:

```
(NP-SBJ
 (NP (NNS Worksheets))
 (PP-LOC
  (IN in)
  (NP
   (NP (DT a) (JJ test-practice) (NN kit))
   (VP
    (VBN called)
    (S
     (NP-SBJ (-NONE- *))
     (NP-PRD-TTL (NNP Learning) (NNPS Materials))))
  (, ,)
  (VP
   (VBN sold)
   (NP (-NONE- *))
   (PP-DTV
    (TO to)
    (NP
     (NP (NNS schools))
     (ADVP-LOC (IN across) (NP (DT the) (NN country))))
    (PP
     (IN by)
     (NP-LGS
      (NNP Macmillan\/McGraw-Hill)
      (NNP School)
      (NNP Publishing)
      (NNP Co.))))
  (, ,))))
)
(S
 (PP-LOC (IN At) (NP (NNP Cray) (NNP Computer)))
 (, ,)
 (NP-SBJ-26 (PRP he))
 (VP
  (MD will)
  (VP
   (VB be)
   (VP
    (VBN paid)
    (NP (-NONE- *-26))
    (NP ($ $) (CD 240,000) (-NONE- *U*))))
  (.
))
```

Can you spot the issue?

According to the transformational grammar formalism of the Treebank, these passive sentences contain an empty node marking the position one of the arguments would occupy in the corresponding active sentence. We should be able to filter these cases by ensuring the NPs we detect are non-empty.
def nonNullNP(tree):
    return tree.label().startswith("NP") and tree[0].label() != "-NONE-

def ppDative(tree):
    for st in tree.subtrees():
        if st.label() == "VP":
            #print("matched vp", st)
            for ch in range(len(st) - 1):
                if (nonNullNP(st[ch]) and st[ch + 1].label() == "PP-DTV"):
                    #print("matched dative", st)
                    return True

def npDative(tree):
    for st in tree.subtrees():
        if st.label() == "VP":
            #print("matched vp", st)
            for ch in range(len(st) - 1):
                if nonNullNP(st[ch]) and nonNullNP(st[ch + 1]):
                    #print("matched dative", st)
                    return True

Does this fix our problem with passives? How are we doing now?

Under the stars and moons of the renovated Indiana Roof ballroom, nine of the hottest chefs in town fed them Indiana duckling mousseline, lobster consomme, veal mignon and chocolate terrine with a raspberry sauce.

Shifted dative
*T*-2 Knowing a tasty -- and free -- meal when they eat one *T*-1, the executives gave the chefs a standing ovation.

Shifted dative
Plans that *T*-13 give advertisers discounts for * maintaining or increasing ad spending have become permanent fixtures at the news weeklies and underscore the fierce competition between Newsweek, Time Warner Inc. 's Time magazine, and Mortimer B. Zuckerman 's U.S. News & World Report.

Shifted dative
The monthly sales have been setting records every month since March.

Shifted dative

Again, the first few look good, but then the next one isn't a dative.

(S
  (NP-SBJ (DT The) (JJ monthly) (NNS sales))
  (VP
    (VBP have)
    (NP
      (VBN been)
      (VP
        (VBG setting)
        (NP (NNS records))
        (NP-TMP
          (NP (DT every) (NN month)))
          (PP (IN since) (NP (NNP March))))))))

Oops! The second NP had better not be some kind of adjunct. I'll exclude NP nodes with a function tag, like NP-TMP:
The lesson so far is that we can tune the program iteratively---we look at what it predicts, check whether we get what we want, and then change things to exclude the false positives.

**False negatives?**

The results look good to me now. Of course, I'm just looking at the sentences which are detected as containing a dative. It's possible the program is failing to detect some actual datives. Obviously we can't look at the entire treebank to figure out whether the detector is missing datives, so we'll have to do something more approximate. Any suggestions?

One possibility is to look at a subset of the data which is likely to have a lot of datives. Let's just inspect sentences using the verb *give*.

Modify your program to run only on sentences for which the verb *give* appears somewhere. (Don't worry about different forms of the verb. The aim is to test our code by extracting some sample sentences which might have datives, not to learn anything about this verb in particular.)

---

**Plans that *T*-13 give advertisers discounts for * maintaining or increasing ad spending**

*have become permanent fixtures at the news weeklies and underscore the fierce competition between Newsweek, Time Warner Inc.'s Time magazine, and Mortimer B. Zuckerman's U.S. News & World Report.*

**Shifted dative**

*In Robert Whiting's *You Gotta Have Wa'*-LRB- Macmillan, 339 pages, $17.95 *U*-RRB-, the Beatles give way to baseball, in the Nipponese version 0 we would be hard put *-2 to call "T"*-1 a "game.""*

**No dative**

*Then, just as an image of the statue of Thomas Jefferson dissolves from the screen, the announcer continues: "On the issue of abortion, Marshall Coleman wants *-1 to take away your right * to choose and *-1 give it to the politicians."

**Prepositional dative**

*By *-3 using them, teachers -- with administrative blessing -- telegraph to students beforehand the precise areas on which a test will concentrate *T*-1, and sometimes give away a few exact questions and answers.*

**No dative**

*The show didn't give the particulars of Mrs. Yeargin's offense, *-1 saying only that she helped students do better on the test.*

**No dative**

*These days, students can often find the answer in test-coaching workbooks and worksheets 0 their teachers give them *T*-1 in the weeks prior to * taking standardized achievement tests.*

**No dative**

*"It's as if France decided *-1 to give only French history questions to students in a European history class, and when everybody ages the test *T*-3, they say 0 their kids are good in European history," says *T*-2 John Cannell, an Albuquerque, N.M., psychiatrist and founder of an educational research organization, Friends for Education, which *T*-105 has studied standardized testing.*

**Prepositional dative**

*The reason: the refusal *ICH*-1 of Congress * to give federal judges a raise.*

**Shifted dative**

*"-2 Founded *-1 by Brooklyn, N.Y., publishing entrepreneur Patricia Poore, Garbage made its debut this fall with the promise * to give consumers the straight scoop on the U.S. waste crisis.*

**Shifted dative**

*He predicted that the board would give the current duo until early next year before * naming a new chief executive.*

**No dative**
What do you think about these cases? Anything suspicious?

What are your feelings on "give way", or on the relative clause structure of "workbooks their teachers give them"? (Notice that the second case actually permits the alternate PP syntax "give to them". But even so, I'm not sure this case is worth searching for.) What about "give until early next year"?

Can you find any more troublesome cases?

### Analysis

Now we've got a dative detector, what do we do with it? We could construct an elaborate study of the dative alternation--- but considering we already did one in the previous module, we'll settle for something simpler: printing the list of verbs that occur with datives and their counts in a neat table.

In order to get this to work, we'll need to come up with a way to find the head verb of each dative construction. Modify your \textit{ppDative} and \textit{npDative} functions so they return this information.

What data structure shall we use to store the counts? How does this relate to speaker genders in Fisher?

If I sort the counts by verbs which take the most dative constructions, the top lines of my table look like this:

<table>
<thead>
<tr>
<th>Verb</th>
<th>Count PP</th>
<th>Probability PP</th>
<th>Count NP</th>
<th>Probability NP</th>
</tr>
</thead>
<tbody>
<tr>
<td>give</td>
<td>31</td>
<td>0.167567567568</td>
<td>154</td>
<td>0.832432432432</td>
</tr>
<tr>
<td>sell</td>
<td>69</td>
<td>0.958333333333</td>
<td>3</td>
<td>0.041666666667</td>
</tr>
<tr>
<td>gives</td>
<td>7</td>
<td>0.112903225806</td>
<td>55</td>
<td>0.887096774194</td>
</tr>
<tr>
<td>gave</td>
<td>17</td>
<td>0.283333333333</td>
<td>43</td>
<td>0.716666666667</td>
</tr>
<tr>
<td>giving</td>
<td>16</td>
<td>0.301886792453</td>
<td>37</td>
<td>0.698113207547</td>
</tr>
<tr>
<td>pay</td>
<td>11</td>
<td>0.297297297297</td>
<td>26</td>
<td>0.702702702703</td>
</tr>
<tr>
<td>sold</td>
<td>29</td>
<td>0.966666666667</td>
<td>1</td>
<td>0.033333333333</td>
</tr>
<tr>
<td>cost</td>
<td>0</td>
<td>0.0</td>
<td>26</td>
<td>1.0</td>
</tr>
<tr>
<td>given</td>
<td>1</td>
<td>0.0434782608696</td>
<td>22</td>
<td>0.95652173913</td>
</tr>
<tr>
<td>paid</td>
<td>8</td>
<td>0.4</td>
<td>12</td>
<td>0.6</td>
</tr>
<tr>
<td>awarded</td>
<td>11</td>
<td>0.611111111111</td>
<td>7</td>
<td>0.388888888889</td>
</tr>
<tr>
<td>sent</td>
<td>7</td>
<td>0.411764705882</td>
<td>10</td>
<td>0.588235294118</td>
</tr>
<tr>
<td>offering</td>
<td>5</td>
<td>0.3125</td>
<td>11</td>
<td>0.6875</td>
</tr>
<tr>
<td>offer</td>
<td>7</td>
<td>0.466666666667</td>
<td>8</td>
<td>0.533333333333</td>
</tr>
<tr>
<td>selling</td>
<td>12</td>
<td>1.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>provide</td>
<td>6</td>
<td>0.545454545455</td>
<td>5</td>
<td>0.454545454545</td>
</tr>
<tr>
<td>offered</td>
<td>3</td>
<td>0.3</td>
<td>7</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Which verbs prefer PP-datatives? Which prefer NP-datatives?