1 Introduction

The aim of the project is to construct classifiers which should assign semantic patterns to six given verbs, as accurately as possible. Each verb has a set of usual usage patterns (based on the Pattern Dictionary of English Verbs), defined by its semantic and syntactic characteristics. It is assumed that if we are able to automatically assign these patterns, it will help us in the general problem of word sense disambiguation which is vital in many fields of computational linguistics.

1.1 The data

The data obtained for this task consist of six text files, one for each verb. Each file contains 250 sentences with the verb, extracted from BNC, with manually annotated pattern tag. In each sentence, the verb in question is highlighted by angle brackets. For each sentence, also a part-of-speech tag list and a dependency structure (both automatically assigned) are included in the file. There is also a named entity list, provided by a Named Entity Recognizer, but this list was not used in this task.

The verbs in question are ally (with 6 semantic patterns), arrive (6 patterns), cry (18 patterns), halt (3 patterns), plough (17 patterns), and submit (5 patterns).

These data represent the training data for the construction of the classifiers. There are 300 more sentences (50 for each verb) for testing the classifiers, but these were not available for the development process, they will be only used for testing and evaluation afterwards.

1.2 Data preprocessing

First what had to be done was to convert the given sentences into feature vectors which could be then processed by the classifiers. For this I created a Perl script which took as its input the text file \{verb\}.txt and outputted a list of feature vectors.

The default set of 275 features\(^1\) was given by the assignment (see Appendix A). Apart from morphosyntactic features that were extracted directly from the POS tagging and dependency structures, there were also semantic features, based on 50 semantic classes extracted from WordNet, which were also given by the assignment.

Most of the features were binary, represented in the output feature list as values “n” and “y”, and a few were categorical, represented by character strings. If needed, the feature script can convert these values to numeric.

The task was then divided in two steps:

---

\(^1\)In fact, the assignment contained 283 features. But since two of the given semantic classes were empty and each semantic class was used to define four features, eight features would necessarily consist of constant zeroes, so I did not implement these eight features.
A. use the default feature set (or its subset), experiment with at least three classification methods, choose the one which seems best and tune it for each of the given verbs.

B. choose three of the six verbs and create new feature lists for them, possibly better than the default features used in step A. Then again, develop a model which will assign the patterns as best as possible.

Apart from the feature extraction from the data, the whole experiment was carried out in the R environment.

2 Baseline for the experiment

As a baseline for the experiment, with which the accuracy of the developed models with be compared, we will use the accuracy of a simple classifier, which assigns to each verb its most frequent pattern tag in the training data. The distribution of patterns differs for different verbs – sometimes there is one pattern which is much more frequent than the others, so even this simple classifier is quite successful.

To get a single number for all six verbs, we will use a weighted average, with weights of verbs defined by their frequency in the BNC. The formula is the following:

\[
\frac{\sum_v p_v \ast A_v}{\sum_v p_v}
\]

where \( p_v \) represents the relative frequency of the verb \( v \), and \( A_v \) the accuracy of the corresponding classifier. The frequencies are shown in Table 1.

<table>
<thead>
<tr>
<th>Verb</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ally</td>
<td>0.0083%</td>
</tr>
<tr>
<td>arrive</td>
<td>0.1307%</td>
</tr>
<tr>
<td>cry</td>
<td>0.0257%</td>
</tr>
<tr>
<td>halt</td>
<td>0.0183%</td>
</tr>
<tr>
<td>plough</td>
<td>0.0076%</td>
</tr>
<tr>
<td>submit</td>
<td>0.0483%</td>
</tr>
<tr>
<td>( \sum_v p_v )</td>
<td>0.2389%</td>
</tr>
</tbody>
</table>

Table 1: Relative frequencies of the verbs

The baseline accuracy computed on all 250 training instances is shown in Table 2.

<table>
<thead>
<tr>
<th>Verb</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ally</td>
<td>47.6%</td>
</tr>
<tr>
<td>arrive</td>
<td>68%</td>
</tr>
<tr>
<td>cry</td>
<td>52.4%</td>
</tr>
<tr>
<td>halt</td>
<td>83.6%</td>
</tr>
<tr>
<td>plough</td>
<td>32.4%</td>
</tr>
<tr>
<td>submit</td>
<td>70.8%</td>
</tr>
<tr>
<td>weighted average</td>
<td><strong>66.2%</strong></td>
</tr>
</tbody>
</table>

Table 2: Baseline accuracy
3 Task A

For this task, I divided the development data for each verb into two parts – 30 sentences were randomly selected as development test data, the rest was used as development train data.

I experimented with three classification methods – Naive Bayes, Decision Trees and Support Vector Machines. For each method, first a specific baseline was computed, which is the accuracy of the classifier with default parameters (i.e. default in the R system) and the full feature set. Next, the tuning process for each method and each verb was the following:

1. from the default set, remove features which have just a single value or have two values, but one of them appears less than four times
2. from the remaining features, select those which improve the accuracy most
3. tune the method’s parameters
4. evaluate the accuracy – by 5-fold cross-validation on the development train data, with a 95% confidence interval. Just for comparison, also the accuracy on development test data was computed.

After the first step, the feature set was reduced significantly:

<table>
<thead>
<tr>
<th></th>
<th>ally</th>
<th>arrive</th>
<th>cry</th>
<th>halt</th>
<th>plough</th>
<th>submit</th>
</tr>
</thead>
<tbody>
<tr>
<td># of features</td>
<td>98</td>
<td>104</td>
<td>111</td>
<td>122</td>
<td>129</td>
<td>107</td>
</tr>
</tbody>
</table>

I decided to select the features first and then tune the parameters, because it seemed plausible that the most useful features will be most useful regardless of the parameters of the method (that parameter tuning might improve absolute performance, but the relative contribution of the features should not change much).

In the following, the accuracy on train set refers to the average accuracy achieved by 5-fold cross-validation on the development train data (220 instances). The accuracy on test set refers to the accuracy on the development test data (the remaining 30 instances).

3.1 Naive Bayes

The baseline accuracy for NB (with the default feature set and default parameters implemented in the R system) is in Table 3:

The selection of a suitable subset of features was done by the greedy algorithm, i.e. starting with an empty set one feature at a time was tested (using 5-fold cross-validation) and the one which improved the performance most was added to the set. In the case of Naive Bayes, two features were taken at the beginning (i.e. all possible pairs were tested at the beginning), because the algorithm implemented in the R system requires at least two features. The selection process was stopped when the overall accuracy stopped increasing significantly, which was usually somewhere around 35 features or earlier (for a graphical representation see Figure 1).

All 220 instances from the development train set were used for training the model in each round.
When the set of features was established, the laplace parameter of the method was tuned – values of 0, 1, 2, and 3 were tested, using 5-fold cross-validation. However, for all the verbs, the value of 0 came best.

Table 4 shows the optimized results:

<table>
<thead>
<tr>
<th>verb</th>
<th># of features</th>
<th>laplace</th>
<th>accuracy on train set</th>
<th>confidence interval</th>
<th>accuracy on test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>ally</td>
<td>32</td>
<td>0</td>
<td>68.2%</td>
<td>± 10.7%</td>
<td>70%</td>
</tr>
<tr>
<td>arrive</td>
<td>19</td>
<td>0</td>
<td>77.3%</td>
<td>± 7.2%</td>
<td>63.3%</td>
</tr>
<tr>
<td>cry</td>
<td>29</td>
<td>0</td>
<td>76.4%</td>
<td>± 6.5%</td>
<td>73.3%</td>
</tr>
<tr>
<td>halt</td>
<td>29</td>
<td>0</td>
<td>88.2%</td>
<td>± 5.4%</td>
<td>86.7%</td>
</tr>
<tr>
<td>plough</td>
<td>33</td>
<td>0</td>
<td>75%</td>
<td>± 8.5%</td>
<td>46.7%</td>
</tr>
<tr>
<td>submit</td>
<td>17</td>
<td>0</td>
<td>90%</td>
<td>± 6.5%</td>
<td>96.7%</td>
</tr>
<tr>
<td>weighted average</td>
<td></td>
<td></td>
<td>80.2%</td>
<td></td>
<td>72.6%</td>
</tr>
</tbody>
</table>

Table 4: Naive Bayes classifier – optimized results

For comparison, I tried also to tune the laplace parameter with all the features, with no prior selection. The results are in Table 5.

<table>
<thead>
<tr>
<th>verb</th>
<th># of features</th>
<th>laplace</th>
<th>accuracy on train set</th>
<th>confidence interval</th>
<th>accuracy on test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>ally</td>
<td>98</td>
<td>2</td>
<td>60%</td>
<td>± 10.1%</td>
<td>60%</td>
</tr>
<tr>
<td>arrive</td>
<td>104</td>
<td>3</td>
<td>68.2%</td>
<td>± 4.9%</td>
<td>76.7%</td>
</tr>
<tr>
<td>cry</td>
<td>111</td>
<td>0</td>
<td>63.2%</td>
<td>± 3.7%</td>
<td>73.3%</td>
</tr>
<tr>
<td>halt</td>
<td>122</td>
<td>3</td>
<td>81.8%</td>
<td>± 9.6%</td>
<td>83.3%</td>
</tr>
<tr>
<td>plough</td>
<td>129</td>
<td>0</td>
<td>63.6%</td>
<td>± 6.9%</td>
<td>56.7%</td>
</tr>
<tr>
<td>submit</td>
<td>107</td>
<td>2</td>
<td>81.4%</td>
<td>± 5.4%</td>
<td>90%</td>
</tr>
<tr>
<td>weighted average</td>
<td></td>
<td></td>
<td>70.9%</td>
<td></td>
<td>78.3%</td>
</tr>
</tbody>
</table>

Table 5: Naive Bayes classifier – tuned results, but no feature selection

The results of the classifier were considerably better, when only some features were selected.
(a) The verb *ally*

(b) The verb *arrive*

(c) The verb *cry*

(d) The verb *halt*

(e) The verb *plough*

(f) The verb *submit*

Figure 1: Feature selection. The dark dot shows where the selection was stopped.
3.2 Decision Trees

Baseline ("non-tuned") accuracy for the Decision Tree classifier is shown in Table 6:

<table>
<thead>
<tr>
<th>verb</th>
<th>accuracy</th>
<th>confidence interval</th>
<th>accuracy</th>
<th>confidence interval</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ally</td>
<td>55%</td>
<td>± 6.7%</td>
<td>63.3%</td>
<td>± 6.7%</td>
<td>63.3%</td>
</tr>
<tr>
<td>arrive</td>
<td>66.8%</td>
<td>± 9.1%</td>
<td>73.3%</td>
<td>± 9.1%</td>
<td>73.3%</td>
</tr>
<tr>
<td>cry</td>
<td>61.8%</td>
<td>± 7%</td>
<td>53.3%</td>
<td>± 7%</td>
<td>53.3%</td>
</tr>
<tr>
<td>halt</td>
<td>83.6%</td>
<td>± 9.4%</td>
<td>83.3%</td>
<td>± 9.4%</td>
<td>83.3%</td>
</tr>
<tr>
<td>plough</td>
<td>58.2%</td>
<td>± 10.9%</td>
<td>50%</td>
<td>± 10.9%</td>
<td>50%</td>
</tr>
<tr>
<td>submit</td>
<td>83.6%</td>
<td>± 5%</td>
<td>90%</td>
<td>± 5%</td>
<td>90%</td>
</tr>
</tbody>
</table>

weighted average 70.3% 74.2%

Table 6: Baseline accuracy for the Decision Tree classifier

With Decision Trees the greedy algorithm did not prove useful for feature selection. It chose a subset of features, but the performance with this subset was not better than the performance achieved when the choice was left to the \texttt{rpart} function implemented in R (i.e. when the function \texttt{rpart} was called with all the features and chose the features for splits by itself).

For the parameter tuning I used the R function \texttt{tune.rpart} and tuned the parameters \texttt{cp} (with values 0.1, 0.05, 0.01, 0.005, and 0.001) and \texttt{minsplit} (with values 2, 4, and 8). The final results are shown in Table 7, the resulting trees are in Figure 2.

<table>
<thead>
<tr>
<th>verb</th>
<th># of features</th>
<th>cp</th>
<th>minsplit</th>
<th>accuracy on train set</th>
<th>confidence interval</th>
<th>accuracy on test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>ally</td>
<td>14</td>
<td>0.01</td>
<td>4</td>
<td>57.7%</td>
<td>± 12.2%</td>
<td>56.7%</td>
</tr>
<tr>
<td>arrive</td>
<td>4</td>
<td>0.05</td>
<td>2</td>
<td>69.5%</td>
<td>± 6.5%</td>
<td>66.7%</td>
</tr>
<tr>
<td>cry</td>
<td>12</td>
<td>0.01</td>
<td>4</td>
<td>65%</td>
<td>± 9.1%</td>
<td>50%</td>
</tr>
<tr>
<td>halt</td>
<td>2</td>
<td>0.05</td>
<td>2</td>
<td>81.8%</td>
<td>± 7.2%</td>
<td>83.3%</td>
</tr>
<tr>
<td>plough</td>
<td>12</td>
<td>0.005</td>
<td>4</td>
<td>60.5%</td>
<td>± 4.3%</td>
<td>63.3%</td>
</tr>
<tr>
<td>submit</td>
<td>7</td>
<td>0.005</td>
<td>8</td>
<td>82.7%</td>
<td>± 8.1%</td>
<td>83.3%</td>
</tr>
</tbody>
</table>

weighted average 71.9% 69.1%

Table 7: Tuned parameters for the Decision Tree classifier

3.3 Support Vector Machines

Similarly to the previous classifiers, first the baseline ("non-tuned") accuracy was computed for the SVM classifier. It is shown in Table 8.

The greedy algorithm for feature selection did not prove useful for the SVM classifier either. Therefore, I used all the features from the default set (apart from the constant or almost constant ones) for parameter tuning.

Parameter tuning was done by the R function \texttt{tune.svm}, and parameters \texttt{gamma} and \texttt{cost} were tuned. Values for \texttt{gamma} were 0.0625, 0.125, 0.25, 0.5, values for \texttt{cost} were 1, 2, 5, 10, 50, 100.

The final results are shown in Table 9.
(a) The verb *ally*  
(b) The verb *arrive*  

(c) The verb *cry*  
(d) The verb *halt*  

(e) The verb *plough*  
(f) The verb *submit*

Figure 2: Decision trees
Table 8: Baseline accuracy for the SVM classifier

<table>
<thead>
<tr>
<th>verb</th>
<th>accuracy on train set</th>
<th>±</th>
<th>confidence interval</th>
<th>accuracy on test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>ally</td>
<td>55.9%</td>
<td>±16.8%</td>
<td></td>
<td>53.3%</td>
</tr>
<tr>
<td>arrive</td>
<td>68.6%</td>
<td>±7.6%</td>
<td></td>
<td>63.3%</td>
</tr>
<tr>
<td>cry</td>
<td>53.2%</td>
<td>±9.3%</td>
<td></td>
<td>46.7%</td>
</tr>
<tr>
<td>halt</td>
<td>83.6%</td>
<td>±6.7%</td>
<td></td>
<td>83.3%</td>
</tr>
<tr>
<td>plough</td>
<td>38.2%</td>
<td>±11.4%</td>
<td></td>
<td>40%</td>
</tr>
<tr>
<td>submit</td>
<td>70.9%</td>
<td>±9.7%</td>
<td></td>
<td>70%</td>
</tr>
<tr>
<td>weighted average</td>
<td><strong>66.9%</strong></td>
<td></td>
<td></td>
<td><strong>63.3%</strong></td>
</tr>
</tbody>
</table>

Table 9: Tuned parameters for the SVM classifier

<table>
<thead>
<tr>
<th>verb</th>
<th>gamma</th>
<th>cost</th>
<th>accuracy on train set</th>
<th>±</th>
<th>confidence interval</th>
<th>accuracy on test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>ally</td>
<td>0.0625</td>
<td>10</td>
<td>64.1%</td>
<td>±15.8%</td>
<td></td>
<td>53.3%</td>
</tr>
<tr>
<td>arrive</td>
<td>0.0625</td>
<td>2</td>
<td>71.4%</td>
<td>±5.9%</td>
<td></td>
<td>66.7%</td>
</tr>
<tr>
<td>cry</td>
<td>0.0625</td>
<td>5</td>
<td>67.3%</td>
<td>±5.1%</td>
<td></td>
<td>73.3%</td>
</tr>
<tr>
<td>halt</td>
<td>0.0625</td>
<td>1</td>
<td>83.6%</td>
<td>±6.7%</td>
<td></td>
<td>83.3%</td>
</tr>
<tr>
<td>plough</td>
<td>0.0625</td>
<td>5</td>
<td>66.8%</td>
<td>±9.1%</td>
<td></td>
<td>66.7%</td>
</tr>
<tr>
<td>submit</td>
<td>0.0625</td>
<td>2</td>
<td>85%</td>
<td>±4.3%</td>
<td></td>
<td>93.3%</td>
</tr>
<tr>
<td>weighted average</td>
<td><strong>74.2%</strong></td>
<td></td>
<td></td>
<td><strong>73.6%</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10: Overall results of Task A

<table>
<thead>
<tr>
<th>method</th>
<th>baseline</th>
<th>tuned performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>66.3%</td>
<td><strong>80.2%</strong></td>
</tr>
<tr>
<td>Decision Trees</td>
<td>70.3%</td>
<td>71.9%</td>
</tr>
<tr>
<td>SVM</td>
<td>66.9%</td>
<td>74.2%</td>
</tr>
</tbody>
</table>

3.4 Results for the Task A

In conclusion to the Task A it can be said that all three methods achieved better performance than baseline. Interestingly, the specific baselines, computed as the performances of the classifiers without tuning any parameters, did not exceed much the primitive baseline, which corresponds to assigning the most frequent pattern to every verb. However, tuning the parameters brought some improvement, and surprisingly the Naive Bayes classifier comes as best. The results are summarized in Table 10.

The classifier, which I chose in the part A of the experiment, is therefore the Naive Bayes classifier, with the laplace parameter set to 0 and with the reduced feature set for each verb (see section 3.1 for details).

Apart from selecting the best possible classifier, we can have a look whether we can learn something interesting about the features from the selections performed with the Naive Bayes classifier and the Decision Tree. Table 11 shows which features were chosen by either of these classifiers and which appeared in both.
If we were to find some general tendency, we see the feature most often considered useful is $f_{274}$ – the presence of a specific prepositional modifier. It appears for all the six verbs in the DT classifier and for three of the verbs in the NB classifier. Among others that appear often are $f_{271}$ – the presence of a phrasal verb particle, $f_{12}$ – nominal following the target verb, and $f_6$ – the VBG tag on the target verb (which means the verb is in gerund or present participle form).

### 4 Comparison with A. Tamchyna’s results

I had the opportunity to compare my results with those of my colleague Aleš Tamchyna [2]. Despite the fact we worked with identical data in the experiment, we obtained different results. To some extent, this was to be expected, because the methods we used are probabilistic in principle and some randomness is always present. But of course I cannot rule out the possibility of an error in my computation.

Our conclusions from the Task A are different: The classifier which comes as best in A. Tamchyna’s experiment, is SVM. In my experiment, it is Naive Bayes. And what are the visible differences in our approaches:

- the number of features removed from the default set – A. Tamchyna removed features that have only one value or their second value appears only once, he ended with cca 140-150 features for each verb. I removed also features, which had their second value less than four times, which yielded cca 100-120 features for each verb. This difference probably does not influence the results very much.
• the ranges of the classifiers’ parameters for tuning – here we differ quite a lot, especially for the DT and SVM classifiers we ended with very different parameters. However, the tuned performance (weighted average) of our classifiers on the development data is quite close:

<table>
<thead>
<tr>
<th></th>
<th>DT classifier</th>
<th>SVM classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>72.9 %</td>
<td>74.6 %</td>
</tr>
<tr>
<td>EK</td>
<td>71.9 %</td>
<td>74.2 %</td>
</tr>
</tbody>
</table>

• the Naive Bayes classifier – there is the biggest difference in the result:

<table>
<thead>
<tr>
<th></th>
<th>NB classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>71.6 %</td>
</tr>
<tr>
<td>EK</td>
<td>80.2 %</td>
</tr>
</tbody>
</table>

There is also probably the biggest difference in approach – I greatly reduced the feature set for this classifier using the greedy algorithm (see section 3.1), for each verb there remained only up to 33 features. This considerably improved the performance. It seems reasonable, because the training data is relatively small, and having too many features is rather a disadvantage, their combinations are not present enough times and classifier is not able to learn them. On the other hand, the greedy algorithm which I used for selecting the most useful features might have the same problem, and it is also possible that the selected features are suitable for our development data, but unsuitable for different data.

5 Comparison with V. Kríž’s results

Vincent Kríž’s diploma thesis is also focused on finding classifiers for automatic pattern assignment, but of course deals with the problem much more thoroughly. He works with more verbs and experiments in much more detail with selecting the best features and tuning the best parameters for each verb and each classification method. For this comparison, I am going to look only on his results for the six verbs I used here and only with morpho-syntactic features (V. Kríž then also experimented with different types of semantic features).

V. Kríž did not use the Naive Bayes classifier, but instead he worked with the k-nearest neighbours classifier and AdaBoost, which I did not. Concerning feature selection, he achieved the best results with a set he calls Best58 – a list of 58 features chosen by a greedy algorithm (for details see [1], pp. 74–75).

The comparison of our results is the following:

<table>
<thead>
<tr>
<th></th>
<th>DT classifier</th>
<th>SVM classifier</th>
<th>NB classifier</th>
<th>kNN classifier</th>
<th>AdaBoost</th>
</tr>
</thead>
<tbody>
<tr>
<td>VK</td>
<td>74.3%</td>
<td>74.8%</td>
<td>x</td>
<td>73.1%</td>
<td>76%</td>
</tr>
<tr>
<td>EK</td>
<td>71.9%</td>
<td>74.2%</td>
<td>80.2%</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

V. Kríž reports the SVM classifier as best. It is because he computes a weighted average over all the 30 verbs he worked with. When computed only over our 6 verbs, it is AdaBoost which gives the best result. However, it is still below the result of Naive Bayes which came out in my experiment.
6 Task B

In the task B, I experimented with adding “dictionary” features, i.e. features of the type “is a word in the neighbourhood of the target verb equal to X?”. I chose the verbs *plough*, *cry*, and *arrive*.

As a starting point, I took the results from the part A. I used the Naive Bayes classifier with a reduced set of features. As the initial set of features for all the verbs I used the features that appeared at least three times in the selections in the part A (see Table 11). These were: f1, f2, f3, f4, f6, f7, f12, f15, f16, f21, f26, f29, f36, f57, f71, f73, and f274. To this set, I added new features for each verb.

The new features were created based on the patterns, which contain a lexical set, e.g. pattern 1 for the verb *plough*:

\[[\text{Human} = \text{Farmer}] \cap [\text{Animal}\{[\text{horse}|\text{ox}|\text{bull}]\}]\] plough (Location | {field|acre|furrow|land|area|moorland|field|fallow|pasture|stetch|round|soil...} = Field)

In some cases, I also added words which were not explicitly mentioned in the pattern, but can logically occur there, e.g. in the pattern 7 for the verb *plough*:

\[[\text{Human} | \text{Institution}] \text{plough} [\text{Money}] [\{\text{into}\} \text{Business Enterprise} | \text{Activity}]\]

Here I replaced the given type [Money] with a set \{money|cash|profit|capital|million|thousand\}.

For descriptions of all the added features see Appendix B.

### 6.1 Results of the Task B

The results of Task B are summarized in Table 12. Even with the initial 17 features only, the classifier performed better than baseline (see Table 2) for the verbs *plough* and *cry*. Adding new features improved the performance for all the three verbs, but not above the level achieved in Task A (shown here in the last column, for details see Table 4).

<table>
<thead>
<tr>
<th>verb</th>
<th>NB with initial features</th>
<th>NB with added features</th>
<th>Task A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>acc. on</td>
<td>conf. acc. on</td>
<td>acc. on</td>
</tr>
<tr>
<td></td>
<td>train set</td>
<td>int. train set</td>
<td>test set</td>
</tr>
<tr>
<td>plough</td>
<td>43.2%</td>
<td>8.2%</td>
<td>46.7%</td>
</tr>
<tr>
<td>cry</td>
<td>58.2%</td>
<td>8.4%</td>
<td>56.7%</td>
</tr>
<tr>
<td>arrive</td>
<td>64.5%</td>
<td>5.5%</td>
<td>56.7%</td>
</tr>
</tbody>
</table>

Table 12: Overall results of Task B

The results of Task B show that “dictionary” features might be of some help. However, it is necessary to select properly the initial set of features, which I did not do. Therefore, the classifiers are not tuned to their best possible performance, I just wanted to experiment with new features to see whether they might be of any use. Also, to find the most suitable dictionary features, we would probably need more development data. It is necessary to have enough examples for each pattern and look carefully into them to extract the words which are most significant for each pattern.
Appendix A

List of Default Features

Syntactic properties of the target verb (TV):

- \( f_0 \) TV in passive voice
- \( f_1 \) modality 1 – TV with would or should
- \( f_2 \) modality 2 – TV with can, could, may, must, ought, might
- \( f_3 \) negation – TV negated

Morphological tags on TV:

- \( f_4 \) VBN tag on TV
- \( f_5 \) VBD tag on TV
- \( f_6 \) VBG tag on TV
- \( f_7 \) VBP tag on TV
- \( f_8 \) VB tag on TV

Properties of the words following and preceding the TV (+- 3 positions):

- \( f_9 \) nominal-like, TV - 3
- \( f_{10} \) nominal-like, TV - 2
- \( f_{11} \) nominal-like, TV - 1
- \( f_{12} \) nominal-like, TV + 1
- \( f_{13} \) nominal-like, TV + 2
- \( f_{14} \) nominal-like, TV + 3
- \( f_{15} \) adjective, TV - 3
- \( f_{16} \) adjective, TV - 2
- \( f_{17} \) adjective, TV - 1
- \( f_{18} \) adjective, TV + 1
- \( f_{19} \) adjective, TV + 2
- \( f_{20} \) adjective, TV + 3
- \( f_{21} \) adverb, TV - 3
- \( f_{22} \) adverb, TV - 2
- \( f_{23} \) adverb, TV - 1
- \( f_{24} \) adverb, TV + 1
- \( f_{25} \) adverb, TV + 2
- \( f_{26} \) adverb, TV + 3

Presence of elements syntactically dependent on TV:

- \( f_{63} \) nominal subject
- \( f_{64} \) clausal subject
- \( f_{65} \) direct object
- \( f_{66} \) indirect object
- \( f_{67} \) passive nominal subject
- \( f_{68} \) passive clausal subject
- \( f_{69} \) clausal complement
- \( f_{70} \) complementizer (“that”, “whether”, etc.)
- \( f_{71} \) any object
- \( f_{72} \) adverbial modifier
- \( f_{73} \) adverbial clause modifier
- \( f_{74} \) purpose clause modifier
- \( f_{75} \) temporal modifier

Semantic classes of the words immediately preceding or following the TV and its subjects and objects:

- \( f_{76} \) TV - 1 class: origin
- \( f_{77} \) TV + 1 class: origin
- \( f_{78} \) subject class: origin
- \( f_{79} \) object class: origin
- \( f_{80} \) TV - 1 class: natural
- \( f_{81} \) TV + 1 class: natural
- \( f_{82} \) subject class: natural
- \( f_{83} \) object class: natural
- \( f_{84} \) TV - 1 class: living
- \( f_{85} \) TV + 1 class: living
- \( f_{86} \) subject class: living
- \( f_{87} \) object class: living
- \( f_{88} \) TV - 1 class: plant
- \( f_{89} \) TV + 1 class: plant
- \( f_{90} \) subject class: plant
- \( f_{91} \) object class: plant
### f0–f270

- **f100** TV -1 class: animal
- **f101** TV -1 class: software
- **f102** subject class: animal
- **f103** object class: animal
- **f104** TV -1 class: artifact
- **f105** TV +1 class: artifact
- **f106** subject class: artifact
- **f107** object class: artifact
- **f108** TV -1 class: form
- **f109** TV +1 class: form
- **f110** subject class: form
- **f111** object class: form
- **f112** TV -1 class: substance
- **f113** TV +1 class: substance
- **f114** subject class: substance
- **f115** object class: substance
- **f116** TV -1 class: solid
- **f117** TV +1 class: solid
- **f118** subject class: solid
- **f119** object class: solid
- **f120** TV -1 class: liquid
- **f121** TV +1 class: liquid
- **f122** subject class: liquid
- **f123** object class: liquid
- **f124** TV -1 class: gas
- **f125** TV +1 class: gas
- **f126** subject class: gas
- **f127** object class: gas
- **f128** TV -1 class: object
- **f129** TV +1 class: object
- **f130** subject class: object
- **f131** object class: object
- **f132** TV -1 class: composition
- **f133** TV +1 class: composition
- **f134** subject class: composition
- **f135** object class: composition
- **f136** TV -1 class: part
- **f137** TV +1 class: part
- **f138** subject class: part
- **f139** object class: part
- **f140** TV -1 class: group
- **f141** TV +1 class: group
- **f142** subject class: group
- **f143** object class: group
- **f144** TV -1 class: function
- **f145** TV +1 class: function
- **f146** subject class: function
- **f147** object class: function
- **f148** TV -1 class: vehicle
- **f149** TV +1 class: vehicle
- **f150** subject class: vehicle
- **f151** object class: vehicle
- **f152** TV -1 class: representation
- **f153** TV +1 class: representation
- **f154** subject class: representation
- **f155** object class: representation

### f268–f274

- **f268** plural subject
- **f269** plural object
- **f270** TV used in an infinite phrase (outside subject)
- **f271** presence of a phrasal verb particle with TV
- **f272** presence of a prepositional modifier
- **f273** presence of a marker (subordination conjunction, apart from “that” and “whether”)
- **f274** presence of a prepositional clausal modifier

---

**Miscellaneous**

- **f268** plural subject
- **f269** plural object
- **f270** TV used in an infinite phrase (outside subject)
- **f271** presence of a phrasal verb particle with TV
- **f272** presence of a prepositional modifier
- **f273** presence of a marker (subordination conjunction, apart from “that” and “whether”)
- **f274** presence of a prepositional clausal modifier

---

All features f0–f270 are binary (yes/no), features f271–f274 are categorical, with the preposition/marker/particle in question as their value.
Appendix B

List of Added Features

plough

- in a window ± 5 words from TV, there is one of the words money, cash, profit, capital, million, thousand or a word containing digits
- among 5 words following TV, there is the word back
- among 6 words following TV, there is the word into
- in a window ± 5 words from TV, there is one of the words field, acre, land, area, pasture, soil, ground
- among 5 words following TV, there is the word sea or ocean
- among 5 words following TV, there is the word path or furrow
- among 3 words following TV, there is the word on, onwards or ahead
- among 3 words following TV, there is the word through

cry

- among 4 words following TV, there is the word eyes, heart or head
- among 6 words following TV, there is the word shoulder
- the word following TV is over
- the word following TV is out
- the word following TV is off
- the word following TV is for
- the word following TV is with
- the word following TV contains the string self or selves

arrive

- in a window ± 6 words from TV, there is one of the words document, collection, exhibition, aid, package, container, letter, cheque, information, mail, goods
- in a window ± 6 words from TV, there is one of the words solution, decision, figure, result, consensus, agreement, answer, interpretation, conclusion
- the word following TV is at
- the word following TV is to or from
- the word following TV is a punctuation mark

References
