Semantic Pattern Classification
Final Report

Aleš Tamchyna *

April 27, 2012

*a.tamchyna@gmail.com

Contents

1 Project Overview

1.1 Semantic Patterns

The goal of the project is to classify occurrences of certain verbs into *semantic patterns*.

Hanks and Pustejovsky (2004) argue that words by themselves do not have a particular meaning. Instead, they have a *meaning potential* with components activated by the word's context. Semantic patterns then correspond to various uses of a word, identified by distinct word context.

Hanks and Pustejovsky (2005) introduce Corpus Pattern Analysis (CPA), a method for acquiring common (*normal*) uses of verbs from corpora—each use corresponds to a certain context pattern. A meaning is associated with each pattern. The authors attempt to cover frequent patterns of verbs in their work in order to achieve a degree of practical usability that discussed alternative approaches lack (WordNet, FrameNet, Levin classes). Unusual uses of verbs are referred to as *exploitations*.

1.2 Training Data

The training data are manually annotated occurrences of selected verbs (*ally, arrive, cry, halt, plough, submit*). The task is to define a good set of features and to develop, evaluate and document supervised classifiers for each verb.

Each verb occurrence is described with rich linguistic information. The context is the whole sentence. For each word, its part of speech and lemma are available. A dependency parse tree of the sentence is given. Finally, the output of named entity recognition is included in the data. An example is given in Figure 1.

```
ID: 577188
PATT: 1
SENT: The scene in the book where Robyn <arrives> at the factory ...
MORPH: The the DT   scene scene NN   in in IN   the the DT   book book NN ...
DEP: det(scene-2, The-1);nsubj(has-19, scene-2);det(book-5, the-4);...
NER: Robyn:P
```

Figure 1: Example of an annotated occurrence of verb *arrive* (simplified).

Before the start (even before feature extraction), I shuffled all sentences to get randomly distributed occurrences of verb patterns.

1.3 Choice of Verbs and Classifiers

Of the allowed classifiers, I selected the following three:

- Naive Bayes

- Decision Trees
- Support Vector Machines

I decided to develop classifiers for verbs *arrive*, *halt*, and *submit*.

2 Classifiers

I give a short theoretical overview of each classifier in this section. This text is based mainly on Mitchell (1997) and the lecture slides[1]. When documenting SVMs, I also used the Wikipedia page[2].

2.1 Naive Bayes

A Bayes learner attempts to model the probability of classifications given the observed feature values. Using this model, we then try to find the most probable classification. More formally, given n features and their values $a_1, \ldots, a_n$ and a set of possible classes $V = \{v_1, \ldots, v_m\}$, we are looking for:

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, ..., a_n)$$

MAP stands for *maximum a posteriori*, i.e. the classification with the largest posterior probability. Using Bayes' Theorem, we rewrite the formula:

$$\begin{aligned} v_{MAP} &= \underset{v_j \in V}{\operatorname{argmax}} \frac{P(a_1, ..., a_n | v_j)}{P(a_1, ..., a_n)} \\ &= \underset{v_j \in V}{\operatorname{argmax}} P(a_1, ..., a_n | v_j) P(v_j) \end{aligned}$$

The prior $P(v_j)$ is easily estimated from the training data by simple counting (we assume the classes in it are distributed identically as in the classified instances). However, the probability of all features given a classification cannot be reliably estimated—we encounter data sparsity unless the feature space is very small or the training data are extremely large.

Naive Bayes classifier makes a drastic assumption of conditional independence of *all* features. We then maximize a simple product of reliably estimated conditional probabilities:

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_{i=1}^{n} P(a_i | v_j)$$

Despite this simplification, naive Bayes classifiers give good results for certain tasks. They are commonly used e.g. in spam filtering systems.

[1] `http://ufal.mff.cuni.cz/~hladka/ML.html`

[2] `http://en.wikipedia.org/wiki/Support_vector_machine`

2.2 Decision Trees

Decision trees are rooted directed trees with varying arity of nodes. Each node corresponds to splitting the data based on value of one particular attribute. Classification is simply "asking questions", i.e. testing the value of an attribute in each node, beginning at root and progressing one level down each time. When a leaf node is reached, the instance is classified (usually) as the most frequent class of the training instances that belong to that leaf node.

One advantage of decision trees is the intuitive simplicity. Also, unlike other learning algorithms, the model can be easily visually inspected and understood (so-called *white-box model*).

Node splits are defined differently for discrete and continuous features. In the first case, a child node is created for each possible attribute value. The same cannot be applied to the latter. One solution for continuous values is to always create two child nodes, setting a c and splitting the instances as follows:

- Instances where $a_i < c$ go to the left child.
- Instances where $a_i \geq c$ go to the right child.

The c value can be found by sorting all instances based on a_i, finding thresholds c_j where the classification changes and selecting among them the one that splits instances into child nodes with the lowest entropy.

Selecting the attribute based on which to split in a particular node can be done in several ways. In all of them, we work with the term *node impurity*, a function that has the maximum value when classes of instances that belong to the node are uniformly distributed and the minimum value when all instances are classified identically. We then select the split that lowers the impurity the most. Strategies for the split selection then differ in the impurity function:

- Misclassification error: $I(t) = 1 - p_t^m$
  p_t^m is the ratio of instances with the most frequent class in the node.
- Information gain: $I(t) = H(t)$
  $H(t)$ is the entropy of classification in the node.
- Gini index: $I(t) = 1 - \sum_i p(i|t)^2$

A decision tree can be constructed using several algorithms. Commonly used is the ID3 algorithm, which is very intuitive. It start with the root node that contains all instances, the node is split based on the splitting criterion and instances are divided among its children based on the value of the selected attribute. ID3 algorithm is then called recursively on each child node.

To avoid overfitting to training data, several techniques can be employed. A parameter can be set that specifies the minimum amount of instances in a node (then the node that

can no longer be split is a leaf node). Pruning can also be used on a fully constructed decision tree—leaf nodes are simply joined together as long as the classification accuracy improves.

2.3 Support Vector Machines (SVM)

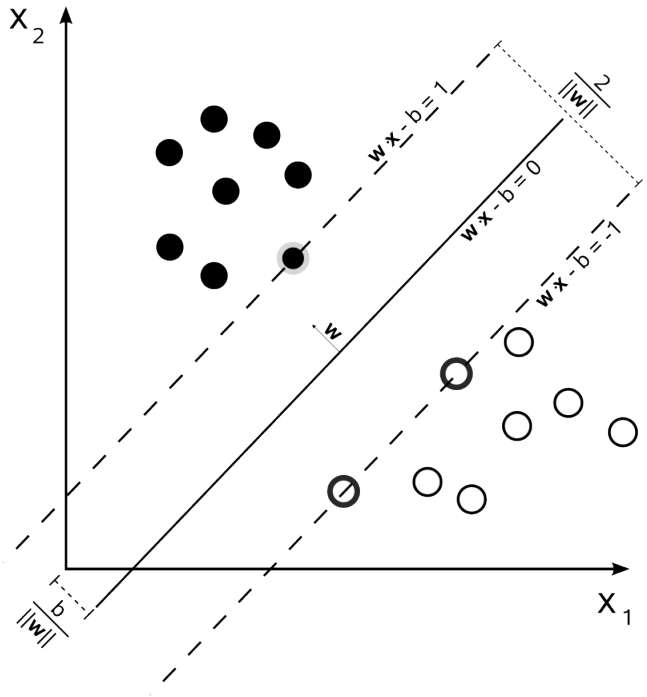

Figure 2: Separating hyperplane and support vectors in SVM.

Support vector machines are a concept used for classification and regression. The basic algorithm only allows for binary classification, however there are several extensions for multi-class SVMs.

Let the training instances be described by n features. SVMs consider the instances to be points in n-dimensional space and attempt to construct a hyperplane that best separates the positive examples from the negative ones:

$$\vec{w}^T \vec{x} + b = 0$$

The hyperplane can be transformed into *canonical form*. In this form, the equations for the closest positive example x_{p_0} and negative example x_{n_0} are, respectively:

$$\vec{w}^T \vec{x_{p_0}} + b = 1$$
$$\vec{w}^T \vec{x_{n_0}} + b = -1$$

These two points are called *support vectors*. Figure 2 illustrates the separating hyperplane and the support vectors associated with it[3].

The best separation is assumed to be the one with the largest margin. This best hyperplane is found by solving a quadratic optimization problem.

Training data are most commonly not linearly separable, i.e. there is no such hyperplane that would separate the data strictly into positive and negative examples—there are outliers, data can be noisy etc. To solve this case, *slack variables* ξ_i are introduced. Originally, classification y_i of a positive example x_i was subject to the following constraint:

$$y_i(\vec{w}, \vec{x_i} + b) \geq 1$$

After adding the slack variables (each training instance has a ξ_i associated with it), the constraint changed:

$$y_i(\vec{w}, \vec{x_i} + b) \geq 1 - \xi_i$$

The optimization problem now takes into account also the slack variables. This technique is also called *soft margins*, referring to the fact that support vectors no longer represent uncrossable borders between negative and positive examples.

The training instances can also be mapped into a higher dimensional space using a kernel function that replaces the *dot product* operation:

$$K : X \times X \to \mathbb{R}$$

Using the *kernel trick* turns SVMs into non-linear classifiers. While there was no good linear separation in the original space, there exists a hyperplane in the transformed feature space that separates the data. Commonly used kernel functions include *polynomial*:

$$K(x_i, x_j) = (x_i \cdot x_j)^d,$$

radial:

$$K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2),$$

and *sigmoid*:

$$K(x_i, x_j) = \tanh(\gamma x_i \cdot x_j - c).$$

2.4 Tunable Parameters in R

Naive Bayes classifier has very few tunable parameters, namely `laplace` and `threshold`, in R terminology. The first parameter is used during training and modifies the behavior of additive smoothing. Setting it to α corresponds to adding α "virtual" occurrences to the

[3]This image is in public domain. Downloaded from:
`http://en.wikipedia.org/wiki/File:Svm_max_sep_hyperplane_with_margin.png`

data. Its default value in R is 0, i.e. no smoothing. The second parameter is an argument of prediction. All zero probabilities in the model are replaced with the given number. Its default value is 10^{-3}.

Decision trees can be tuned in various ways. I experimented with varying values of the `cp`, complexity parameter and `minsplit`. Both parameters limit the maximum size of the resulting decision tree. The *complexity parameter* puts a lower bound on the improvement of impurity achieved by a split – if the impurity is not lowered enough, the node will not be split. The second parameter disallows splitting nodes that contain less data instances than its value.

Support vector machines have numerous parameters. From the described classifier types, they are also the most difficult to tune. The user can select the `kernel` function (linear = no transformation, polynomial, radial and sigmoid). Each of these kernels has specific "hyperparameters":

- Linear—none, simple scalar product.
- Polynomial—`gamma`, `degree`, `coef0`.
- Radial—`gamma`.
- Sigmoid—`gamma`, `coef0`.

Finally, the `cost` parameter sets the penalty for misclassified training instances. An exhaustive search over all parameter combinations is not feasible—my approach to tuning loosely followed the recommendations described in Hsu et al. (2003).

3 Feature Set

3.1 Default Features

The default feature set is described in the project specification.

- Morpho-syntactic features.
  - Characteristics of the verb (10 binary features).
  - Characteristics of verb neighbours (54 binary features).
  - Characteristics of verb syntactic dependents (15 binary and 4 categorical features).
    - Logical subjects.
    - Objects.
    - Particles.
    - Adverbials.
- Semantic features (4×50 = 200 binary features).

3.2 Feature Selection

Many of the default features are activated only rarely. I experimented with filtering of features that were active less than k times. Motivation for this is twofold—these features help models to overfit on the training data and they slow down the training and classification. I found that filtering features that were active never or once does no harm. Surprisingly, filtering less rare features (up to 5 occurrences) caused the classification accuracy on the test data to slightly decrease. I decided to filter features that occurred less than twice always as a pre-processing step. Note that the remaining features are different for each verb. I used this feature set for Task A.

3.3 Alternative Feature Sets

The following two feature sets were evaluated in Task B.

3.3.1 Parts of Speech of Close Words

Aside from the rich default feature set, I also tried a much simpler and shorter description of the context—for each verb occurrence, there are 6 categorical features denoting the POS tags of its surrounding words (three preceding and three that follow it[4]). If the verb is near sentence beginning or end, I assign a special tag `<s>` to the exceeding positions. All punctuation is denoted as `punct`, for technical reasons.

Although the features are categorical, SVMs implicitly convert the features to binary flags for each possible category, allowing for a geometric interpretation.

3.3.2 Part of Speech Groups

This feature set is similar to the previous one with an exception—the POS tags are grouped in the same way as was suggested for the default feature set.

Similarly to the previous feature set, SVM implicitly converts each categorical feature into a set of binary features.

4 Classifier Tuning

For each classifier type, the tuning procedure was as follows:

1. Split off last 50 instances as *test set*.
2. For each tested combination:
   (a) Split data into 5 equally-sized blocks.
   (b) Evaluate the accuracy using 5-fold cross-validation.

[4]Other context lengths had lower accuracy.

3. Train a classifier on the whole training data
   using the best parameter combination.

4. Evaluate the accuracy on the test data.

Note that all data were shuffled beforehand. In SVMs, I used 5-fold cross-validation that is built in the `svm()` function.

5 Task A

Tuning of parameters was done via a grid search over empirically selected sets of possible parameter values. It is possible that for some verbs/classifiers a better parameter value might be either out of range of the tested values, or somewhere between two values. Accuracies for each parameter combination evaluated in the grid search are documented in Appendix A.

I used the default feature set with the simple filtering described in Section 3.2. Filtering of the default features resulted in different subsets for each verb. On the other hand, only features occurring zero times or once were filtered out, so the impact on results is negligible.

Verb	ally	arrive	cry	halt	plough	submit
Feature Count	140	149	147	151	155	137

Table 1: Number of features remaining after filtering.

The following table summarizes baseline accuracies (baseline assigns the most frequent pattern to all instances).

Verb	ally	arrive	cry	halt	plough	submit
Accuracy	0.476	0.68	0.524	0.836	0.324	0.708

Table 2: Baseline accuracies.

Regarding the tuning of SVM machines, I used the radial kernel function recommended in Hsu et al. (2003). A short evaluation of other kernel types confirmed that this function is the most hopeful.

I tried tuning all classifiers on all verbs. See Table 3 for details. 95% confidence interval is given for each accuracy value. With respect to the final scoring function, SVMs seemed to perform best. I therefore chose them as the classifier for Task A.

Using Welch paired two-sample t-test on accuracies from the cross-validation, I checked whether the difference between the SVM classifiers and the baseline is significant. For 3 verbs, the difference was statistically significant. For *cry*, it would be if α was 0.1. The verb *halt* had a very strong baseline, so the difference is far from significant. In the case *arrive*, the SVM results varied considerably during cross-validation.

	ally	arrive	cry	halt	plough	submit
Naive Bayes	0.62±0.077	0.7±0.091	0.605±0.034	0.825±0.116	0.55±0.126	0.82±0.102
Decision Tree	0.605±0.102	0.71±0.052	0.595±0.092	**0.83±0.115**	0.495±0.077	**0.87±0.060**
SVM	**0.66±0.148**	**0.725±0.062**	**0.67±0.129**	**0.83±0.040**	**0.595±0.123**	0.85±0.054

Table 3: Development set accuracies of classifiers.

Classifier	Weighted Accuracy
Naive Bayes	0.716±0.090
Decision Tree	0.729±0.065
SVM	**0.746±0.082**

Table 4: Weighted accuracies of classifiers.

	ally	arrive	cry	halt	plough	submit
Cost	10	100	10	100	10	100
Gamma	0.071	0.067	0.007	0.001	0.065	0.001

Table 5: SVM parameters.

ally	arrive	cry	halt	plough	submit
0.58	0.8	0.72	0.92	0.64	0.82

Table 6: Test set accuracies of the best classifiers.

	ally	arrive	cry	halt	plough	submit
P-value	0.040	0.208	0.083	0.716	0.005	0.001
Significant	+	-	-	-	+	+

Table 7: Results of significance tests, $\alpha = 0.05$.

6 Task B

I did not create features for any verb in particular. Instead, I evaluated the usability of 3 different (universal) feature sets (see Section 3.2, Section 3.3.1 and Section 3.3.2). Results on the default (filtered) feature set are discussed in the previous section.

6.1 POS of Close Words

This feature set was too complex for decision trees—the number of possible values for each feature overwhelmed the training algorithm, which seemed to run indefinitely. For this

reason, only naive Bayes classifier and SVMs are evaluated in this section.

The following table summarizes evaluation results. The only classifier significantly better than the baseline is printed in bold. Again, 95% confidence interval is stated for each value.

Verb	arrive	halt	submit
Naive Bayes	0.66±0.071	0.815±0.109	**0.77±0.060**
SVM	0.695±0.060	0.815±0.071	0.79±0.113

Table 8: Development set accuracies.

6.2 POS Groups of Close Words

I evaluated performance of all types of classifiers for each verb. The achieved results are summarized in the following tables. 95% confidence interval is stated for each value.

Verb	arrive	halt	submit
Naive Bayes	0.675±0.062	0.82±0.080	0.725±0.031
Decision Tree	0.665±0.064	0.815±0.092	0.71±0.078
SVM	0.67±0.119	0.815±0.071	0.735±0.084

Table 9: Development set accuracies.

All classifier types perform very similarly—this feature set is not very challenging, so this is not surprising. Unfortunately, all results are similar to or worse than with previous feature sets and none of them are significantly better than the baseline (measured the same way as in the previous sections).

6.3 Comparison of Feature Sets

To select the best classifier for each verb, I compare the best classifiers for each verb from the feature sets.

	arrive	halt	submit
Default	**0.725** (SVM)	**0.83** (SVM)	**0.87** (DT)
POS	0.695 (NB)	0.815 (NB/SVM)	0.79 (SVM)
POS Group	0.675 (NB)	0.82 (NB)	0.735 (SVM)

Table 10: Accuracies of the best classifiers in each feature set. The best overall are in bold.

Apparently the initial impression that a simpler feature set would be beneficial was wrong—any simplification past the filtering of sparse features from the default set harms accuracy.

References

P Hanks and J Pustejovsky. A pattern dictionary for natural language processing. *Revue Française de Linguistique Appliquée*, 10(2):63–82, 2005.

Patrick Hanks and James Pustejovsky. Common sense about word meaning: Sense in context. In Petr Sojka, Ivan Kopecek, and Karel Pala, editors, *TSD*, volume 3206 of *Lecture Notes in Computer Science*, pages 15–18. Springer, 2004. ISBN 3-540-23049-1.

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification, October 29 2003. URL `http://citeseer.ist.psu.edu/689242.html; http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

A User Documentation

A.1 Project Files

`best-a.r`	Evaluate Task A classifiers on a test set.
`best-b.r`	Evaluate Task B classifiers on a test set.
`eval.r`	Calculate weighted score on Task A.
`extract_default_features.pl`	Extract default feature set.
`extract_pos_features.pl`	Extract POS categorical features.
`extract_pos_group_features.pl`	Extract POS-group features.
`filter_columns.pl`	Remove given columns from data set.
`filter_features.pl`	Filter rare features.
`final.r`	R code for tuning, cross-validation.
`tuning.default`	Log file of tuning of default (filtered) set.
`tuning.pos`	Log file of tuning of POS feature set.
`tuning.pos_groups`	Log file of tuning of POS-group feature set.

A.2 Usage

To evaluate the classifiers on a new test set, run the following command pipeline:

```
$ for verb in ally arrive cry halt plough submit; do \
    cat <your_test_file> | ./extract_default_features.pl \
    | ./filter_columns.pl $(cat data/$verb.discarded) \
    > data/$verb.test \
done

$ R
> source("best-a.r")
> source("best-b.r")
```

B Default Feature Set—Tuning Accuracies

The following tables document the grid search for best parameter combination. Standard deviation based on 5-fold cross-validation is given for each measurement. Bold font denotes the best achieved accuracy.

The "Gamma" in the tables does not denote the actual value of the `gamma` parameter. Instead, it is the numerator of the following fraction:

$$\texttt{gamma} = \frac{Gamma}{\text{number of features}}$$

The default value for `gamma` in R is 1/(number of features).

Threshold \ Laplace	0	1	2	3	4
0.1	0.38±0.021	0.575±0.040	0.555±0.067	0.555±0.054	0.545±0.054
0.01	0.56±0.049	0.575±0.040	0.555±0.067	0.555±0.054	0.545±0.054
0.001	**0.62±0.062**	0.575±0.040	0.555±0.067	0.555±0.054	0.545±0.054
0.0001	0.6±0.053	0.575±0.040	0.555±0.067	0.555±0.054	0.545±0.054

Table 11: *ally*, default feature set: Naive Bayes development set accuracies.

CP \ Minsplit	0	5	10	20	40
0.1	0.52±0.078	0.52±0.078	0.52±0.078	0.52±0.078	0.52±0.078
0.01	**0.605±0.082**	0.6±0.11	0.57±0.093	0.575±0.077	0.55±0.035
0.001	0.59±0.08	0.575±0.094	0.56±0.091	0.575±0.077	0.54±0.042
0.0001	0.59±0.08	0.575±0.094	0.56±0.091	0.575±0.077	0.54±0.042

Table 12: *ally*, default feature set: Decision tree development set accuracies.

Cost \ Gamma	0.01	0.1	1	5	10
0.1	0.455±0.093	0.455±0.093	0.455±0.093	0.455±0.093	0.455±0.093
1	0.455±0.093	0.455±0.093	0.54±0.096	0.57±0.11	0.585±0.14
10	0.455±0.093	0.55±0.10	0.605±0.13	0.655±0.12	**0.66±0.12**
100	0.55±0.10	0.61±0.12	0.595±0.11	0.63±0.12	0.635±0.12
1000	0.61±0.12	0.605±0.08	0.575±0.12	0.63±0.12	0.635±0.12

Table 13: *ally*, default feature set: SVM radial kernel development set accuracies.

Threshold \ Laplace	0	1	2	3	4
0.1	0.475±0.047	0.675±0.025	**0.69±0.058**	0.665±0.029	0.66±0.042
0.01	0.67±0.045	0.675±0.025	0.69±0.058	0.665±0.029	0.66±0.042
0.001	0.69±0.065	0.675±0.025	0.69±0.058	0.665±0.029	0.66±0.042
0.0001	0.7±0.073	0.675±0.025	0.69±0.058	0.665±0.029	0.66±0.042

Table 14: *arrive*, default feature set: Naive Bayes development set accuracies.

CP \ Minsplit	0	5	10	20	40
0.1	0.635±0.029	0.635±0.029	0.635±0.029	0.635±0.029	0.635±0.029
0.01	0.685±0.042	0.67±0.033	0.675±0.053	0.685±0.049	**0.71±0.042**
0.001	0.625±0.018	0.67±0.021	0.665±0.068	0.68±0.041	0.71±0.042
0.0001	0.625±0.018	0.67±0.021	0.665±0.068	0.68±0.041	0.71±0.042

Table 15: *arrive*, default feature set: Decision tree development set accuracies.

Cost \ Gamma	0.01	0.1	1	5	10
0.1	0.665±0.10	0.665±0.10	0.665±0.10	0.665±0.10	0.665±0.10
1	0.665±0.10	0.665±0.10	0.665±0.10	0.67±0.094	0.675±0.098
10	0.665±0.10	0.665±0.10	0.705±0.09	0.7±0.061	0.72±0.054
100	0.665±0.10	0.7±0.095	0.68±0.087	0.695±0.074	**0.725±0.05**
1000	0.705±0.10	0.68±0.089	0.685±0.08	0.695±0.074	0.725±0.05

Table 16: *arrive*, default feature set: SVM radial kernel development set accuracies.

Threshold \ Laplace	0	1	2	3	4
0.1	0.41±0.076	0.575±0.064	0.58±0.045	0.57±0.074	0.55±0.077
0.01	0.565±0.055	0.575±0.064	0.58±0.045	0.57±0.074	0.55±0.077
0.001	**0.605±0.027**	0.575±0.064	0.58±0.045	0.57±0.074	0.55±0.077
0.0001	0.6±0.031	0.575±0.064	0.58±0.045	0.57±0.074	0.55±0.077

Table 17: *cry*, default feature set: Naive Bayes development set accuracies.

Minsplit / CP	0	5	10	20	40
0.1	0.585±0.095	0.585±0.095	0.585±0.095	0.585±0.095	0.585±0.095
0.01	0.575±0.064	0.585±0.058	0.57±0.06	**0.595±0.074**	0.56±0.055
0.001	0.56±0.07	0.585±0.065	0.54±0.09	0.58±0.072	0.565±0.063
0.0001	0.56±0.07	0.585±0.065	0.54±0.09	0.58±0.072	0.565±0.063

Table 18: *cry*, default feature set: Decision tree development set accuracies.

Gamma / Cost	0.01	0.1	1	5	10
0.1	0.52±0.074	0.52±0.074	0.52±0.074	0.52±0.074	0.52±0.074
1	0.52±0.074	0.52±0.074	0.52±0.074	0.625±0.059	0.64±0.065
10	0.52±0.074	0.52±0.074	**0.67±0.10**	0.67±0.11	0.67±0.089
100	0.52±0.074	0.655±0.11	0.635±0.11	0.66±0.098	0.665±0.086
1000	0.655±0.11	0.595±0.086	0.58±0.045	0.66±0.098	0.665±0.086

Table 19: *cry*, default feature set: SVM radial kernel development set accuracies.

Laplace / Threshold	0	1	2	3	4
0.1	0.495±0.096	0.77±0.11	**0.825±0.094**	0.81±0.084	0.815±0.088
0.01	0.68±0.11	0.77±0.11	0.825±0.094	0.81±0.084	0.815±0.088
0.001	0.72±0.089	0.77±0.11	0.825±0.094	0.81±0.084	0.815±0.088
0.0001	0.75±0.040	0.77±0.11	0.825±0.094	0.81±0.084	0.815±0.088

Table 20: *halt*, default feature set: Naive Bayes development set accuracies.

Minsplit / CP	0	5	10	20	40
0.1	0.805±0.086	0.805±0.086	0.805±0.086	0.81±0.076	0.815±0.074
0.01	0.75±0.11	0.735±0.11	0.8±0.12	**0.83±0.093**	0.77±0.07
0.001	0.75±0.11	0.735±0.11	0.8±0.12	0.83±0.093	0.77±0.07
0.0001	0.75±0.11	0.735±0.11	0.8±0.12	0.83±0.093	0.77±0.07

Table 21: *halt*, default feature set: Decision tree development set accuracies.

Cost \ Gamma	0.01	0.1	1	5	10
0.1	0.815±0.058	0.815±0.058	0.815±0.058	0.815±0.058	0.815±0.058
1	0.815±0.058	0.815±0.058	0.815±0.058	0.815±0.058	0.815±0.058
10	0.815±0.058	0.815±0.058	0.825±0.043	0.815±0.068	0.805±0.082
100	0.815±0.058	**0.83±0.033**	0.795±0.057	0.795±0.074	0.805±0.078
1000	0.83±0.033	0.805±0.048	0.795±0.057	0.795±0.074	0.805±0.078

Table 22: *halt*, default feature set: SVM radial kernel development set accuracies.

Threshold \ Laplace	0	1	2	3	4
0.1	0.37±0.086	0.475±0.087	0.47±0.054	0.485±0.042	0.46±0.029
0.01	0.51±0.11	0.475±0.087	0.47±0.054	0.485±0.042	0.46±0.029
0.001	**0.55±0.10**	0.475±0.087	0.47±0.054	0.485±0.042	0.46±0.029
0.0001	0.545±0.096	0.475±0.087	0.47±0.054	0.485±0.042	0.46±0.029

Table 23: *plough*, default feature set: Naive Bayes development set accuracies.

CP \ Minsplit	0	5	10	20	40
0.1	0.315±0.038	0.315±0.038	0.315±0.038	0.315±0.038	0.315±0.038
0.01	**0.495±0.062**	0.495±0.062	0.49±0.052	0.455±0.072	0.415±0.058
0.001	0.475±0.040	0.455±0.074	0.48±0.074	0.455±0.072	0.415±0.058
0.0001	0.475±0.040	0.455±0.074	0.48±0.074	0.455±0.072	0.415±0.058

Table 24: *plough*, default feature set: Decision tree development set accuracies.

Cost \ Gamma	0.01	0.1	1	5	10
0.1	0.345±0.093	0.345±0.093	0.345±0.093	0.345±0.093	0.345±0.093
1	0.345±0.093	0.345±0.093	0.345±0.093	0.5±0.085	0.515±0.082
10	0.345±0.093	0.355±0.10	0.555±0.086	0.585±0.068	**0.595±0.10**
100	0.355±0.10	0.55±0.073	0.585±0.049	0.58±0.097	0.59±0.096
1000	0.555±0.076	0.58±0.054	0.59±0.058	0.58±0.097	0.59±0.096

Table 25: *plough*, default feature set: SVM radial kernel development set accuracies.

Threshold \ Laplace	0	1	2	3	4
0.1	0.465±0.088	**0.82±0.082**	0.815±0.063	0.755±0.048	0.735±0.029
0.01	0.605±0.10	0.82±0.082	0.815±0.063	0.755±0.048	0.735±0.029
0.001	0.67±0.074	0.82±0.082	0.815±0.063	0.755±0.048	0.735±0.029
0.0001	0.695±0.082	0.82±0.082	0.815±0.063	0.755±0.048	0.735±0.029

Table 26: *submit*, default feature set: Naive Bayes development set accuracies.

CP \ Minsplit	0	5	10	20	40
0.1	0.82±0.033	0.82±0.033	0.82±0.033	0.82±0.033	0.82±0.033
0.01	0.82±0.054	0.855±0.041	0.855±0.045	**0.87±0.048**	0.84±0.052
0.001	0.805±0.062	0.855±0.041	0.855±0.045	0.87±0.048	0.84±0.052
0.0001	0.805±0.062	0.855±0.041	0.855±0.045	0.87±0.048	0.84±0.052

Table 27: *submit*, default feature set: Decision tree development set accuracies.

Cost \ Gamma	0.01	0.1	1	5	10
0.1	0.705±0.065	0.705±0.065	0.705±0.065	0.705±0.065	0.705±0.065
1	0.705±0.065	0.705±0.065	0.705±0.065	0.82±0.089	0.82±0.089
10	0.705±0.065	0.705±0.065	0.84±0.052	0.84±0.06	0.845±0.06
100	0.705±0.065	**0.85±0.043**	0.815±0.058	0.83±0.06	0.845±0.067
1000	0.85±0.043	0.805±0.065	0.805±0.06	0.83±0.06	0.845±0.067

Table 28: *submit*, default feature set: SVM radial kernel development set accuracies.