Semantic type classification
PFL054 (Introduction to Machine Learning) Project

Anna Lauschmannová
anna.lauschmannova@gmail.com

February 20, 2011

1 Project description

The aim of this project is to create a binary classifier for distinguishing between ordinary and semantic collocations. By a *collocation* we mean any meaningful and grammatical word combination that often occurs in natural language. The meaning and usage of many collocations may be inferred directly from the meaning of its parts. On the other hand, *semantic collocations* are multiword expressions that are lexically, syntactically, semantically, pragmatically and/or statistically idiosyncratic. It means that semantic collocations have semantic and/or syntactic properties that cannot be fully predicted from those of their components, and therefore semantic collocations have to be listed in a lexicon.

For example, compare the following 5 expressions: "they are", "black horse", "red wine", "black market", and "weapons of mass destruction". While the first two are considered only "ordinary" collocations, the other three are semantic collocations.

2 Data description

The available data consists of 9232 collocation candidates extracted from the [PDT 2.0][1].

[1]The data was taken from the set of the collocation candidates occurring in the PDT at least six times and having part-of-speech patterns that can possibly form a collocation. The total size of this set is 12 232 candidates — the remaining 3000 candidates will be used for the evaluation of this project.

2.1 Baseline hypothesis

1945 of these candidates are semantic collocations, the rest are not. Thus, with the simples hypothesis that no candidates are semantic collocations, we get the *baseline accuracy* of 78.9%[2].

This hypothesis is more prone to errors resulting from biased sampling of the population than the models obtained by learning. To demonstrate this, let us think of the well-known *iris data set* [And35]. It consists of 50 instances of each three different species of the iris flower. 50 instances represent a large enough sample for determining the variance of the features within each species as well as differences between them. However, it is extremely unlikely that the proportion of these species would be $1 : 1 : 1$ in any real life situation. If we want to distinguish Iris setoza from the other two species, the proportion of train instances leads us to the hypothesis that no instances are setoza. The accuracy of the hypothesis on the training data is 66%. However, if we take a sample from a geographical location rich in setoza, the accuracy will be much worse — and if we take a sample from a location where setoza does not grow, the accuracy may be 100%. Therefore, if we were to classify unknown samples of iris flowers, we would prefer any model based on features to the model based on proportion in the training data, even if its accuracy on the training data would be much below 66%.

Nonetheless, in the case of semantic collocations, we will assume that if a new sample is obtained with the same preprocessing and filtering of the candidates, the ratio of semantic collocations among the test instances will be roughly the same as in the training data. It would be interesting to investigate how far is this assumption misleading when we start handling texts from different domains, genres or languages.

2.2 Features

There are 100 features assigned to each one of the candidates; they are listed in Tables 1 and 2.[3] The feature named `tp` is the classification — true or false semantic collocation; there are also three features, named `a, b` and `z`, which have been obtained from human annotators and should not be used for the classification as they directly determine the feature `tp`.

Six features are categorial: the lemma, part-of-speech and dependency relation for each of the two words.

The remaining ninety features take numeric values. It is important to realize that for some machine learning algorithms, it is necessary to scale these values so that all of the features have the same variance.

[2] In this report, all accuracies are rounded to one per mille.

[3] See [Pec09, p. 39–44] for a more detailed description of the lexical association measures.

Table 1: List of basic features

	feature name	feature description
	Categorial features	
l1	Lemma 1	Lemma of the first word.[5]
t1	POS tag 1	Reduced part-of-speech tag of the first word.[6]
a1	Dependency relation 1	Value "Head" if the first word is a head of the bigram, simplified dependency type otherwise.
l2	Lemma 2	Lemma of the second word.[5]
t2	POS tag 2	Reduced Part-Of-Speech tag of the second word.[6]
a2	Dependency relation 2	Value "Head" if the second word is a head of the bigram, simplified dependency type otherwise.
	Frequency features	
fA	frequency	Frequency of the dependency bigram.
fB	second without first	Frequency of the second word not being in a dependency relation with the first word.
fC	first without second	Frequency of the first word not being in a dependency relation with the second word.
fD	other bigrams	Frequency of all other dependency bigrams in the PDT.
	Occurence statistics	
b1	Mean gap number	
b2	Variance gap number	
b3	Mean component offset	
b4	Variance component offset	
	Multiword expression (MWE) categories	
a	first annotator	**0:** not a semantic collocation; **1:** stock phrases, frequent unpredictable usages; **2:** names of persons, organizations, geographical locations, and other entities; **3:** support verb constructions; **4:** technical terms; **5:** idiomatic expressions
b	second annotator	
z	third annotator	
tp	semantic collocation	**0** for false; **1** for true semantic collocation (all three annotators assigned a number different from 0)

[5]By "lemma" we generally mean the "lemma proper" (without technical suffixes) of PDT 2.0, see section 2.1. "Lemma structure" of the "Manual" or `http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/en/m-layer/html/ch02s01.html`.

[6]By part-of-speach we mean the concatenation of the 1st, 3rd, 10th, and 11th character of the PDT 2.0 morphological tag (positional tag), see section 2.2.1.1. "Part of speech" of the "Manual" or `http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/en/m-layer/html/ch02s02s01.html#POS`.

Table 2: List of lexical association measures

Association measures
`x1` Joint probability; `x2` Conditional probability; `x3` Reverse conditional probability; `x4` Pointwise mutual information; `x5` Mutual dependency; `x6` Log frequency biased mutual dependency; `x7` Normalised expectation; `x8` Mutual expectation; `x9` Salience; `x10` Pearson's χ^2 test; `x11` Fisher's exact test; `x12` Student's t test; `x13` z score; `x14` Poison significance measure; `x15` Log likelihood ratio; `x16` Squared log likelihood ratio
Association coefficients
`x17` Russel-Rao; `x18` Sokal-Michiner; `x19` Rogers-Tanimoto; `x20` Hamann; `x21` Third Sokal-Sneath; `x22` Jaccard; `x23` First Kulczynski; `x24` Second Sokal-Sneath; `x25` Second Kulczynski; `x26` Fourth Sokal-Sneath; `x27` Odds ratio; `x28` Yulle's ω; `x29` Yulle's Q; `x30` Driver-Kroeber; `x31` Fifth Sokal-Sneath; `x32` Pearson; `x33` Baroni-Urbani; `x34` Braun-Blanquet; `x35` Simpson; `x36` Michael; `x37` Mountford; `x38` Fager; `x39` Unigram subtuples; `x40` U cost; `x41` S cost; `x42` R cost; `x43` T cost; `x44` Phi; `x45` Kappa; `x46` J measure; `x47` Gini Index; `x48` Confidence; `x49` Laplace; `x50` Conviction; `x51` Piatersky-Shapiro; `x52` Certainity factor; `x53` Added value; `x54` Collective strength; `x55` Klosgen
Context measures
`x56` Context entropy; `x57` Left context phrasal entropy; `x58` Right context phrasal entropy; `x59` Left context divergence; `x60` Right context divergence; `x61` Cross entropy; `x62` Reverse cross entropy; `x63` Intersection measure; `x64` Euclidean norm; `x65` Cosine norm; `x66` $L1$ norm; `x67` Confusion probability; `x68` Reverse confusion Probability; `x69` Jensen-Shannon divergence; `x70` Cosine of pointwise mutual information; `x71` KL divergence; `x72` Reverse KL divergence; `x73` Skew divergence; `x74` Reverse skew divergence; `x75` n-gram word coocurrence; `x76` n-gram word assotiation
Context similarity
`x77` Cosine context similarity in boolean vector space; `x78` Cosine context similarity in *term frequency* vector space; `x79` Cosine context similarity in *term frequency–document frequency* vector space; `x80` Dice context similarity in boolean vector space; `x81` Dice context similarity in *term frequency* vector space; `x82` Dice context similarity in *term frequency–document frequency* vector space

In order to get some preliminary idea about the features, I plotted several kinds of plots. In all plots throughout this report, red points represent semantic collocations and grey points represent ordinary collocations. This is achieved by the following commands:[4]

```
colors <- rep("grey35",number_of_instances)
colors[data_all[,"tp"]==1] <- "red"
```

We may start with the investigation of each feature and its relationship to the classification `tp`. This may be done in three different ways:

(a) plotting the selected feature against feature `tp`:

```
plot(data[,feature], data[,"tp"], col=colors)
```

(b) plotting the value of the feature against the instance number:

```
plot(data[,feature], 1:number_of_instances, col=colors)
```

If the order of the instances is random, the points in this kind of plot are spread over a large area, and hence more of them are actually visible.

(c) plotting the cummulative graph:

```
colors <- rep("grey35",number_of_instances)
colors[data[,"tp"][names(sort(data[,feature]))]==1] <- "red"
plot(sort(data[,feature]), 1:number_of_instances, col=colors)
```

Figure 1 shows a sample of these plots for features `x2, x11` and `x18`. From the first kind of plot we see immediately that feature `x2` by itself will not be a good predictor of `tp`; the other two features seem far more interesting, as for each one of them there is an interval $I \subseteq \mathbb{R}$ such that if for any given instance x the value of the feature is in I, then x is classified as ordinary collocation.[7]

The first kind of plot for features `a1` and `a2` also shows that predicates, auxiliaries, appositions, complements of non-verb elements and nominal parts of predicates with copula never appear in semantic collocations (at least in our data). This information could be used to extend the pre-processing of the data.

The second and third kind of plot give us a better idea about the distribution of the values. We find out that although all semantic collocations have `x11` very near 0 and larger values imply "ordinary" collocation, it helps us to classify only few instances — simply almost all instances have `x11` near 0. On the other hand,

[4]Throughout this report, I add snippets of the R code when I consider it interesting. However, the code is simplified, for example I do not list all optional arguments.

[7]However, no single feature f completely predicts `tp` in the sense that we could find an $r \in \mathbb{R}$ such that if $f(x) < r$ then x is a semantic collocation and otherwise it is not — or vice versa.

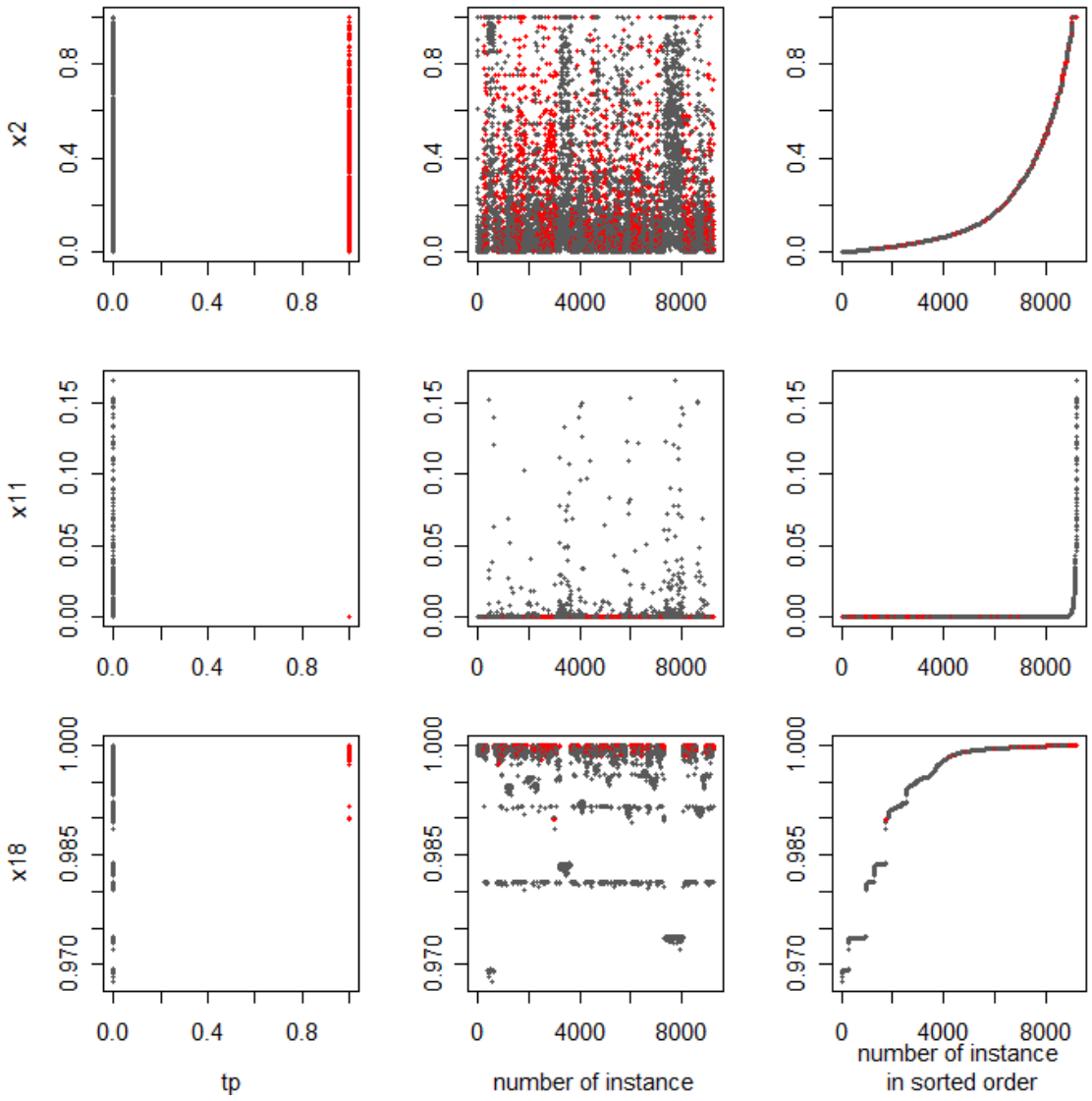

Figure 1: Three kinds of plots for features `x2, x11` and `x18`

`x18` allows a larger set of values for the positive instances, but about twice as many instances fall into the interval I containing only negative instances.

We also notice that while features `x2` and `x11` have a continuous range of values, the values of `x18` form at least three disjoint intervals. We may ask ourselves whether this corresponds to any natural characteristics of the data, such as the part-of-speech. This leads us directly to the idea of plotting pairs of features agains each other. By plotting `x18` against `a1, a2, t1` and `t2` we find out that these "clusters" correspond neither to the part-of-speech of one of the members nor to

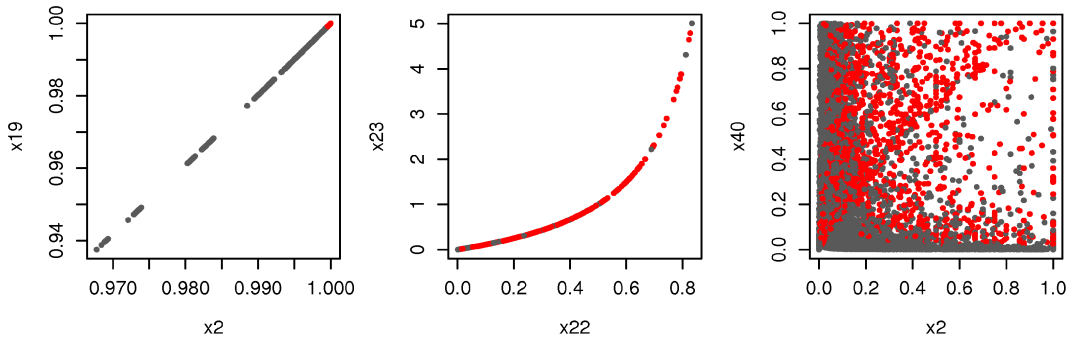

Figure 2: a) Linearly correlated features `x18` and `x19`; b) non-linearly correlated features `x22` and `x23`; c) uncorrelated features `x2` and `x40`

the dependency relation between them.

When we plot the features against each other, we can see that some of them are strongly (often almost linearly) correlated (Figure 2).

The use of several linearly correlated features is similar to the use of uncorrelated but weighted features; in the second approach it is more obvious which features are "preferred". Nonetheless, I have used the whole feature set for my experiments.

2.3 Collocation candidates

It should be noted here that instances of the co-occurence of the same pair of words are further distinguished as to the dependency relation between them and the detailed part-of-speech of both members. Thus, for example, the word-pair "zcela jiný" is considered as three different candidates: in all three cases, "jiný" is the Head and it is an adjective, but it is either feminine, neuter or masculine inanimate. Due to the use of the detailed part-of-speech tags, these three cases form separate entries in the data.

Among 8531 different word pairs that appear in the data, there are 633 word pairs that correspond to more than one candidate, covering altogether more than 14% of the candidate set (1334 candidates). However, there are only 7 word pairs such that some corresponding candidate is a semantic collocation and some is not (see Table 3).

From our data it seems that it is highly unlikely that the same word pair would be used in one gramatical context only in its literal meaning and in other gramatical

Table 3: Word pairs such that some and not all of the corresponding candidates are semantic collocations

l1	a1	t1	l2	a2	t2	a	b	z	tp
československý	Atr	AF1A	armáda	Head	NF-A	1	1	2	1
československý	Atr	AX1A	armáda	Head	NF-A	1	0	2	0
doba	Head	NF-A	určitý	Atr	AF1A	4	1	4	1
doba	Head	NF-A	určitý	Atr	AF1N	0	4	0	0
konec	Head	NI-A	rok	Adv	NI-A	1	1	1	1
konec	Head	NI-A	rok	Atr	NI-A	1	0	1	0
konec	Head	NI-A	rok	Adv	NN-A	0	0	1	0
tak-3	Adv	D—	zvaný	Head	AF1A	4	4	1	1
tak-3	Adv	D—	zvaný	Head	AI1A	4	4	0	0
uzavřít	Head	V—	dohoda	Obj	NF-A	3	3	3	1
uzavřít	Head	V—	dohoda	Sb	NF-A	0	0	3	0
velký	Atr	AF1A	společnost	Head	NF-A	1	1	1	1
velký	Atr	AF3A	společnost	Head	NF-A	0	0	1	0
vyspělý	Atr	AF1A	země	Head	NF-A	4	4	4	1
vyspělý	Atr	AF3A	země	Head	NF-A	0	0	1	0

context as a semantic collocate.[8] The question whether this is linguistically plausible or rather result of the annotation scheme (see [Pec09] for the details) should be left to linguists. However, it seems reasonable to use the word pair as the primary key for the classification of new instances. By this I mean the following strategy: when faced with a new instance, first check whether the same word pair has appeared in one of the training candidates. If so, and if it is not one of the 7 "undecidable" cases, assign to it the same value of the `tp` feature as seen in the training data. Only if the word pair has not been seen in the training data, use your favourite model for the prediction.

2.4 Development and test data

I divided the data into two parts, the development part consisting of 8260 instances and the "publication" part intended for final evaluation of the selected methods consisting of 972 instances. The development part was further divided into 7

[8]Note that the annotators were instructed to judge any candidate which could eventually appear in a context where it has character of collocation as true collocation [Pec09, p. 56]. This includes the possibility that the same candidate (with given grammatical structure) sometimes appears as a semantic collocate and sometimes in its literal meaning. However, a difference may arise when the same word pair appears with different grammatical structure. This is the case for seven word pairs in Table 3.

subparts with 1180 instances each which I used for cross-validation during the parameters optimization.

3 Theoretical aspects of methods (models)

3.1 k-Nearest Neighbours

In this method, a new instance is assigned the classification which is most common among k training instances which are nearest to it. If there is no clear winner, the algorithm selects at random. Clearly, the concept of "nearest" neighbours depends on the metric that is used; the most common option is the Euclidean metric, in which case it is impossible to use categorial features. However, categorial features may be converted to numerical ones for example by creating a new numerical feature for each possible value of the categorial feature:

$$f(x) \in I \quad \longrightarrow \quad \forall i \in I \quad f_i(x) = 1 \text{ iff } f(x) = i.$$

Scaling and weighing is especially important for the k-NN algorithm because it strongly influences the distances between the points. Figure 3 demonstrates this. In this toy example, I used six training points: "s-1 funkce", "všechen firma", "z-1 území", "židovský stát-1", "živnostenský úřad", "životní podmínka"; the last three are semantic collocations. The new instance "zájemce muset" is then classified

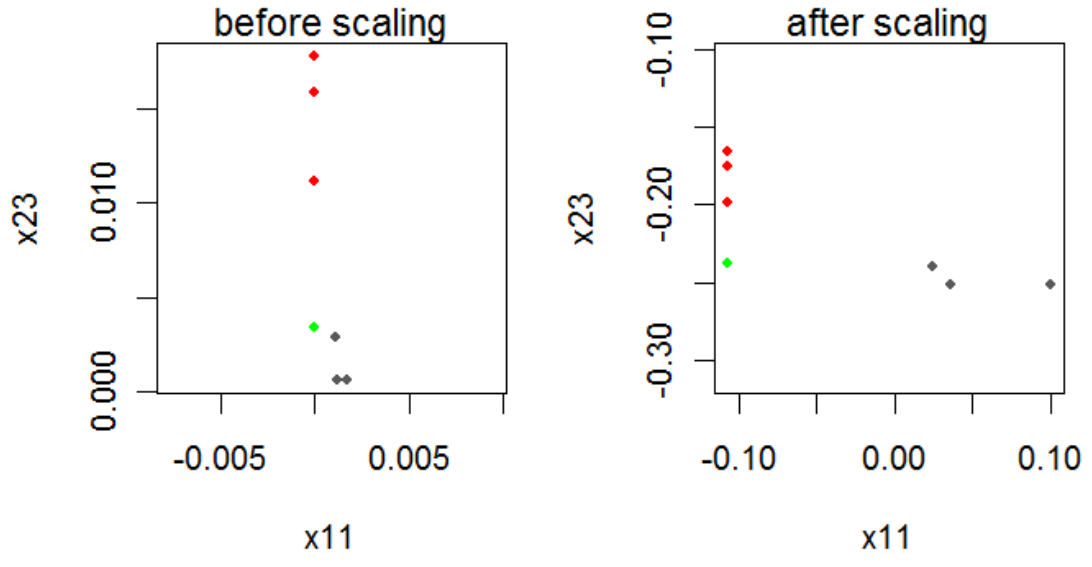

Figure 3: The effect of scaling on the choice of nearest neighbours

using 3 nearest neighbours and feature space consisting of features `x11` and `x23`; before scaling[9] it is classified as ordinary collocation but after scaling it is classified as semantic collocation. We see that before scaling, feature `x23` has a larger impact on the classification because the values of the feature `x11` almost do not differ; however, during the scaling, `x11` was multiplied by about 118, whilst `x23` was only multiplied by roughly 5. This means that the distances on the `x11` axis have increased much more than the distances on the `x23` axis, or in yet another words, `x11` was given a larger weight.

3.2 Decision trees and boosting

As the second method, I selected decision trees with boosting.

Decision trees as such use a very intuitive approach: as we have already mentioned, a simple look at the plots of some features shows that there is an interval I such that if the value of the feature falls into this interval, the predicted function is known. (See Figure 1.) If we were to classify some new instances manually, we would definitely start with something like this:

Look at the dependency relations `a1` and `a2`; one of them is Head, look at the other one:

1. if it is Atv, Aux, Pnom, Pred or Apos, classify the candidate as ordinary collocation;
2. else look at feature `x18`:
   (a) `x18` < 0.985, classify the candidate as ordinary collocation;
   (b) else ...

After a while we would arrive at a set of training instances which cannot be clearly cut with the help of any feature, i.e. for each feature f and each $x \in \mathbb{R}$, neither $f(i) < x$ nor $f(i) > x$ implies a definite classification of the instance i. In the decision tree approach, this is not considered a serious obstacle. Some feature and some subset of its values are selected so that the division of instances into two subsets according to $f(i)$ creates subsets which are less mixed with respect to the predicted function. The subsets are then divided with the help of further features; the process goes on recursively. Only when such recursively constructed subset becomes smaller than a certain preset number, the most common class is assigned

[9] I scaled the features to have mean 0 and variance 1 on the whole data, with no respect to the instances selected for this example. The mean value is not important for the k-NN algorithm, it is just a side effect of the `scale` function in R, which first centralizes each column and then scales it.

to all instances in the subset, creating some small error, but avoiding overfitting to the training data.

More exact discussion of decision trees may be found on [Wikipedia] in articles "Decision tree learning", "ID3 algorithm", "C4.5 algorithm" and "Decision tree pruning", or [Hla10, Decision trees].

Boosting is a general method for developing classifiers with high accuracy on the training data. The main idea of the algorithm is to iteratively train models on the same data, giving higher weight to the training instances that were classified incorrectly in the previous iteration step. Finally, all models are combined into a single one using weighted voting. Because the instances that were classified incorrectly in one iteration are given higher weight in the next iterations, boosting leads to improved accuracy on the training data. See [Fre99] for more details.

3.3 Naive Bayes classifier

As the third method, I selected the Naive Bayes classifier. It is a direct implementation of the Bayes' rule for prediction based on several pairwise independent features:

$$\texttt{tp}_{PREDICT} = \text{argmax}_{\texttt{tp}\in\{0,1\}} P(\texttt{tp}) \prod_{i=1}^{96} P(a_i|\texttt{tp}),$$

where $a_i, i \in \{1, \ldots 96\}$ are the values of the features for the given instance. Besides the assumption that the features are independent (which is made for any Naive Bayes classifier), the implementation in R also assumes that the continuous features have Gaussian distribution given the target class. As we have seen in the discussion of features in Section 2.2, neither of these two assumptions holds for this data. And indeed, of the three kinds of models that I have experimented with, the Naive Bayes classifier gave the poorest results.

4 Implementation of the selected methods

My best classifier can be quickly obtained by running the script

Lauschmannova.best.model.R;

it is also in the file

Lauschmannova.RData

as the object `LAUSCHMANNOVA_BEST_MODEL`.

4.1 k-Nearest Neighbours

The k-Nearest Neighbours algorithm is implemented in the R system in several different ways. Below I list them with the most commonly used parameters and their short explanations (not the default values).

- library `class` for basic prediction and cross-validation:

```
prediction <- knn(train = train_data, test = test_data,
                  cl = train_classes,
                  k = num_of_nearest_neighbours,
                  l = minimum_num_of_votes_otherwise_doubt,
                  prob = return_proportion_of_votes?,
                  use.all = use_all_inst_of_same_distance?)
prediction <- knn.cv(train, cl, k, l, prob, use.all)
```

  The latter has the advantage that the data does not need to be divided into training and test part, but the cross-validation is performed by the *take-one-out* method on the whole training data; this is equivalent to size-of-the-data–fold crossvalidation.

- library `kknn` for weighed k-NN:

```
kknn(class ~ ., train = train_data, test = test_data,
     k = num_of_nearest_neighbours,
     distance = parameter_of_Minkowski_distance,
     kernel = "rectangular"_Euklidean___"triangular"___
              _"inv"_1/distance___"gaussian"___others)
train.kknn(class ~ ., data, distance, kernel,
           kmax = try_all_k_in_1:kmax)
```

  Both of these functions return a complex object with many components. One of them is the list of the fitted values (in `train.kknn` for every combination of k and kernel). The former of these two functions also returns classes, weights and distances of the k nearest neighbours. The latter allows for automatic evaluation of misclassification erros and best parameters; and – most importantly – it already employs the leave-one-out cross-validation.

- library `knnflex` for adjusting the way the distance is calculated:

```
distances <- knn.dist(train_and_test_data,
         dist.meth = any_of_euclidean_maximum_manhattan_
```

```
                          canberra_binary_minkowski,
          p = power_of_minkowski_distance)
knn.predict(train = train_rows_of_data,
            test = test_rows_of_data,
            dist.matrix = distances_from_knn.dist,
            k = num_nearest_neighbours,
            ties.meth = "min"_use_all___"max"_use_none___
                        "random"___"first"_in_data)
knn.probability(train, test, dist.matrix, k, ties.meth)
```

- library `RWeka`

```
classifier <- IBk(class ~ .,
                  data,
                  subset, na.action,
                  control = Weka_control(
                              I = weight_by_1/distance?,
                              F = weight_by_1-distance?,
                              K = num_nearest_neighbours,
                              X = take_one_out_cv_for_k_in_1:K
                                       )
                 )
summary(classifier)
evaluate_Weka_classifier(classifier,
                numFolds = num_of_folds_for_crossvalidation,
                newdata = for_evaluation_on_test_set)
predict(classifier, test_data_set)
```

For the selection of the parameters, I decided to use `train.kknn`, because it uses the leave-one-out cross-validation and it outputs the best combination of parameters. As far as I could tell, `IBk` was somewhat quicker, but I came across a problem with the interpretation of the evaluation on previously unseen data which I could not solve, so I rather used `train.kknn`.[10]

[10]The problem was the following: when I used the function `evaluate_Weka_classifier` with the parameter `newdata`, the confusion matrix looked differently than when I constructed the confusion matrix with `table(newdata[,"tp"], predict(classifier, newdata))`. Because I could not explain this difference, I was not sure about the right interpretation of the output of `RWeka` functions and decided not to use them.

4.2 Decision trees and boosting

I used the `ada` function from the package of the same name. The following piece of code stores the accuracies on the train and test part of the development data and can be used for later decision which combination of parameters works best and how many iterations of the boosting algorithm should be performed:

```
control <- rpart.control(method="class", minsplit=min, cp=cpcko,
                         maxdepth=max, maxsurrogate=0, xval=7)
dec_tree <- ada(tp~., data=data_train[,c(-1, -4)],
                test.x=data_test[,c(-1, -4, -97)],
                test.y=data_test[,97],
                type = "gentle", control=control, iter=30)
dec_tree_accuracies[[paste("minsplit=",min, ", cp=", cpcko,
                           ", maxdepth=", max, sep=" ")]] <-
                    dec_tree$model$err
```

Note that in the decision tree approach, it is necessary to exclude the two features containing the lemmas, because otherwise the algorithm would tend to build a decision tree based on them.

The `rpart` function already contains some cross-validation, so I decided not to use cross-validation for every possible combination of parameters, as this would be too time consuming: the boosting algorithm constructs a large number of decision trees and if the boosting itself should be performed several times in order to cross-validate, it would take too long. After going through a larger set of possible combinations of the parameters `minsplit, cp` and `maxdepth`, I selected three of them by their accuracy on one, fixed portion of the development data (called `data_test` in the above code). Only for these three combinations I performed cross-validation, dividing the development data into `data_test` and `data_train` in seven different ways and selecting the combination that gave the best mean accuracy on `data_test`. However, I am aware that the confidence intervals for some of these mean accuracies overlap.

4.3 Naive Bayes classifier

Again, this is a method which is sensitive to the choice of particular features. I experimented with the ordering of the features that can be obtained by the package `FSelector`. With the help of the functions in this package, it is easy to order the features according to the `information.gain`, `gain.ratio`, `chi.squared` coefficient or `symmetrical.uncertainty` relative to feature `tp`. However, the functions in FSelector assume that the features are discrete, so applying them to continuous features might be extremely misleading. In selecting the parameters for

the Naive Bayes algorithm, I decided to use the following strategy: going through the features starting with those marked by FSelector as most relevant, and for each feature, adding it to the feature set if this leads to improved accuracy of the classifier. I experimented with allowing the same feature to be added several times, although this violates the independence assumption present in the Naive Bayes classifier model.

After some experiments, the best accuracy still was only 83.3% (with features `x4+x4+x4+x4+x37+x50+fA`); as this is still much less than what I could obtain with decision trees or the k-NN algorithm, so I did not try to further optimize tha parameters for the Naive Bayes classifier.

5 Project analysis

After choosing one combination of parameters for each of the three methods, I performed seven-fold cross-validation[11] and a test on the "publication" data for the three models. For the test on the "publication" data I used the whole development set as the training set. The confidence intervals are computed for the 95% confidence level. A comparison of the results may be found in Table 4.[12]

A few things need to be said about the table:

The performance of the best classifier on the unseen data is even higher than the upper bound of the confidence interval. This may be pure chance or it may be because the last classifier was trained on a larger training set. An argument in favour of the latter explanation would be that also the accuracy of the k-NN algorithm on the publication data is at the upper end of the confidence interval. However, it may simply be the case that by chance the publication set is such that it is easy to predict it from the development set and some other division would behave differently.

The confidence intervals do not overlap at all. Thus we may confidently say that of these three classifier, the boosted model is the best and the Naive Bayes classifier is only slightly better than the baseline. And indeed, we know from the

[11] In this final evaluation, I performed crossvalidation with the same division of the data into folds for all three classifiers; during development, I used crossvalidation methods specific to the models, i.e. the "leave-one-out" cross-validation for k-Nearest Neighbours, the built-in cross-validation of the function `ada` and the usual cross-validation with the Naive Bayes classifier. Each one of these is suitable for choosing the best combination of parameters for a given method; however, their results cannot really be compared.

[12] Just for fun, let me note that if the models are trained on the (unseen) publication data and evaluated on the development data (on which they were tuned), their respective accuracies are 86.5%, 89.1% and 82.1%. These numbers are smaller than the accuracy on unseen data in the table, probably because they are obtained from training on a much smaller set.

Table 4: Chosen parameters and resulting accuracies

method R library R function	parameters	mean accuracy	conf. interval (lower, upper)	accuracy on unseen data

k-Nearest Neighbours
 	
library `kknn`
`kknn`
	kernel: "inv"
	k: 18
	44 features:
	`x2+x3+x5+x7+x10+x12+x13`
	`+x20+x21+x22+x23+x24+x27`
	`+x29+x30+x33+x34+x37+x38`
	`+x39+x41+x45+x47+x48+x51`
	`+x53+x55+x57+x59+x60+x61`
	`+x63+x64+x65+x69+x70+x74`
	`+x77+x80+x81+fA+fC+b3+b4`
		87.6	86.9, 88.3	88.3

decision trees with boosting library `ada` `ada`	minsplit: 5 cp: -1 maxdepth: 7 iter: 21	89.9	89.4, 90.4	90.5
Naive Bayes library `e1071` `naiveBayes`	features: `x4+x4+x4+x4+x37+x50+fA` laplace: 0.001	82.1	81.4, 82.8	82.0

theory that boosting yields increased performance already after a small number of iterations.

However, the poor performance of the Bayes classifier is probably caused by poor parameter tuning, not by insufficiency of the method itself.

6 What else to learn

Here, I would like to mention a few topics that I believe I need to consider seriously in my future attempts to use machine learning methods.

6.1 Feature selection

One of the major issues in tuning both k-NN and Naive Bayes classifier is the selection of features. I need to learn more about measures of feature relevancy to

the predicted function, and about the R functions that can be used to determine such measures. In particular, I did not find any suitable package for working with continuous features predicting a discrete function and I suspect that the functions `chi.squared`, `info.gain`, `gain.ratio` and `symmetrical.uncertainty` from the `FSelector` package give misleading predictions by assuming that the features are discrete. On the other hand, `linear.correlation` did not seem relevant for a two-class problem.

I also assume that different measures would be relevant for different models: for the k-NN algorithm, a measure of feature relevance should have something to do with distances between points; for the Bayesian approach, such a measure will likely be based on some statistical theory.

Also, it has been suggested in the literature that simpler machine learning algorithms may be used for selection of features which are later used in more complex algorithms: for example, we might first evaluate misclassification error of the k-NN algorithm with each feature separately (for some fixed k) and then select features with smallest error; or we might decide to work only with those features that are actually used in a decision tree built from the whole data. Both of these methods are also implemented in the `FSelector` package, however, they are rather time consuming.

6.2 Optimization algorithms

I have also found out that I need to learn more about optimization strategies and algorithms. The only method really covered in the class was "try many values from the parameter space and choose the best one". However, the size of the parameter space is growing exponentially with increasing number of parameters and/or their possible values.

References

[And35] Edgar Anderson (1935). "The irises of the Gaspé Peninsula". Bulletin of the American Iris Society 59: 2–5. The Iris flower data set was first used as an example of discriminant analysis in [Fis36]. The complete Iris flower data set can be found at `http://en.wikipedia.org/wiki/Iris_flower_data_set` or within the "HSAUR" package of the R-system.

[Fis36] Fisher, R.A. (1936). "The Use of Multiple Measurements in Taxonomic Problems". Annals of Eugenics 7: 179–188. `http://digital.library.adelaide.edu.au/coll/special//fisher/138.pdf`.

[Fre99] Yoav Freund, Robert E. Schapire: "A Short Introduction to Boosting". Journal of Japanese Society for Artificial Intelligence, 14(5), p. 771-780, September 1999. Translation by Naoki Abe. Available online at `http://www.site.uottawa.ca/~stan/csi5387/boost-tut-ppr.pdf`.

[Hla10] Barbora Vidová Hladká, teaching materials for the course Introduction to machine learning (in natural language processing), academic year 2010/11; chapter "Decision trees" is available at `http://ufal.mff.cuni.cz/~hladka/jsmath/test/decision-tree-learning.pdf`.

[PDT 2.0] Prague Dependency Treebank 2.0, Linguistic Data Consortium, catalog number LDC2006T01. See `http://ufal.mff.cuni.cz/pdt2.0/`.

[Pec09] Pavel Pecina: Lexical Association Measures: Collocation Extraction. In the series Jan Hajič (ed.). Studies in Computational and Theoretical Linguistics. Institute of Formal and Applied Linguistics, Prague, Czech Republic, 2009.

[Wikipedia] English Wikipedia, `http://en.wikipedia.org/wiki/`.

Contents

List of Tables

List of Figures