11 Classification Algorithm and Feature Combination Techniques

An advantage of token-classification approaches is that they are not coupled to one specific statistical method. *Any* classification algorithm that can return a probability distribution over the active classes can be used as the core of such approaches. In our case, the modular design of our system makes it easy to replace the classifier (it is not even necessary to recompile the rest of the system).

So when we have to decide on a classification algorithm to be used in our system, this only concerns the choice of a *default* classifier; any other classifiers can still be integrated as alternatives. Since one of our goals to to allow *incremental training* to reduce the human training effort, it makes sense to chose an *online learning algorithm* as default classifier. Online algorithms learn by processing each of the training examples in the order they come in; they do not need access to the whole training corpus at once (as many other algorithms do, for example for feature selection).

The online learning algorithm we will use as default classifier is *Winnow*. Winnow is specifically suitable if the overall number of features is very high, since it is a simple and fast algorithm that can be (and, in our case, is) implemented using a sparse architecture where the time required for training or classification depends only on the number of features that are active (present) in the current example, while the overall number of features known to the classifier is irrelevant. In spite of being simple, Winnow still gets very competitive results (as we will see in the evaluation part, cf. esp. Chap. 16), especially when used with one of the feature combination techniques we will introduce in Section 11.2.

11.1 The Winnow Classification Algorithm

The Winnow algorithm introduced by [Lit88] is a statistical, but not a probabilistic algorithm, i.e. it does not directly calculate probabilities for classes. Instead it calculates a *score* for each class.

Our variant of Winnow is suitable for both binary (two-class) and multi-class (three or more classes) classification. It keeps an n-dimensional weight vector $w^c = (w_1^c, w_2^c, \ldots, w_n^c)$ for each class c, where w_i^c is the weight of the ith feature. The algorithm returns 1 for a class iff the summed weights of all active features (called the score Ω) surpass a predefined threshold θ:

$$\Omega = \sum_{j=1}^{n_a} w_j^c > \theta.$$

Otherwise ($\Omega \leq \theta$) the algorithm returns 0. $n_a \leq n$ is the number of active (present) features in the instance to classify.

The goal of the algorithm is to learn a linear separator over the feature space that returns 1 for the true class of each instance and 0 for all other classes on this instance. The initial weight of each feature is 1.0. The weights of a class are updated whenever the value returned for this class is wrong. If 0 is returned instead of 1, the weights of all active features are increased by multiplying them with a *promotion factor* α, $\alpha > 1$:

$$w_j^c \leftarrow \alpha \times w_j^c.$$

If 1 is returned instead of 0, the active weights are multiplied with a *demotion factor* β, $0 < \beta < 1$:

$$w_j^c \leftarrow \beta \times w_j^c.$$

The classification tasks we need to address in our approach do not yield feature vectors of a fixed size. In text classification, the number of features depends on the length of the text, so it can vary enormously from instance to instance. Similarly, the size of the context representations we generate as feature vectors for information extraction (cf. Sec. 12.2) varies depending on the position of a token in the current document and the size of surrounding elements such as the sentence it is in. Thus instead of using a fixed threshold we set the threshold to the number n_a of features that are active in the given instance: $\theta = n_a$. Thus initial scores are equal to θ since the initial weight of each feature is 1.0.

In multi-label classification, where an instance can belong to several classes at once, the algorithm would predict all classes whose score is higher than the threshold. But for our purposes, there is exactly one correct class for each instance, thus we employ a *winner-takes-all* approach where the class with the highest score is predicted.

This means that there are situations where the algorithm will be trained even though it did not make a mistake. This happens whenever the scores of both classes[1] are at the same side of the threshold and the score of the true class is higher than the other one—in this case the prediction of Winnow will be correct but it will still promote/demote the weights of the class that was at the wrong side of the threshold.

Analogously to SNoW (cf. [Car04, p. 20]), we convert the scores returned by a Winnow instance into confidence (probability) estimates using the function

$$p(\Omega^c, \theta) = \frac{e^{(\Omega^c - \theta)/\theta}}{\sum_{c' \in C} e^{(\Omega^{c'} - \theta)/\theta}},$$

where $Omega^c$ is the score for class c (from the set of all classes C) and θ is the threshold as explained above (the denominator is the normalization factor).

The complexity of processing an instance depends only on the number of active features n_a, not on the number of all features n_t. We use a sparse architecture where features are allocated whenever the need to promote/demote them arises for the first time. In sparse Winnow, the number of instances required to learn a linear separator (if

[1] of two or more classes in tasks involving more than two classes

exists) depends linearly on the number of relevant features n_r and only logarithmically on the number of active features, i.e. it scales with $O(n_r \log n_a)$ (cf. [Mun99, Sec. 2]).

Winnow is a non-parametric approach; it does not assume a particular probabilistic model underlying the training data, nor does it require features to be independent of each other.

11.1.1 Thick Threshold

In our implementation of Winnow, we use a *thick threshold* for learning (cf. [Dag97, Sec. 4.2]). Instances are trained even if the classification was correct if the determined score was near the threshold. Two additional thresholds θ^+ and θ^- with $\theta^- < \theta < \theta^+$ are defined and each instance whose score falls in the range $[\theta^-, \theta^+]$ is considered a mistake. In this way, a large margin classifier will be trained that is more robust when classifying borderline instances.

11.1.2 Classification between Multiple Classes

While our Winnow variant supports multi-class classification, initial experiments indicated that is advantageous to use multiple binary classifiers in a "one-against-the-rest" setup. We train a separate classifier for each of the classes defined by the employed tagging strategy (cf. Sec. 10.2), except for the background class (the O class according to the used naming schemas). The context representations of all tokens of a non-background class are trained as positive instances for the corresponding classifier and as negative instances for all other classifiers; the context representations of background tokens are trained as negative instances for all classifiers.

In the application stage, we compare the predictions of the separate classifiers. If several classifiers predict their positive class, the most confident classifiers wins. If none of the positive classes is chosen, the background class (O) wins.

11.1.3 Incremental Training and Batch Training

An advantage of Winnow is its supporting *incremental* (online) training as well as *batch* (iterative) training. The advantages of incremental training have already been discussed (Sec. 3.3). Typical trainable IE systems require to be batch-trained from a set of annotated training texts. The resulting statistical model can be used to propose extractions from other (similar) texts, but it cannot be changed without being rebuilt from scratch. However, for many "real-life" applications, automatic extractions will be checked and corrected by a human revisor, as automatically extracted data will always contain errors and gaps that can be detected by human judgment only. This correction process continually provides additional training data, but batch-trainable algorithms are not very suited to integrate new data, since full retraining is required which will take increasingly longer times.

On the other hand, batch training generally leads to superior results on static corpora where training and testing documents are randomly drawn from the same set.

Being an online algorithm, Winnow naturally supports incremental training. Batch-training Winnow is possible by repeating this online training process several times. For batch training, we repeatedly train the classification model on the whole training set until one of the following conditions is true:

1. Classification accuracy on the training set did not increase compared to the last iteration;
2. Classification accuracy on the training set is already perfect (0 errors); or
3. The algorithm has been trained for 15 iterations.

The last condition (which is configurable) is to avoid overtraining. In our experiments, it generally was irrelevant, since the batch training process tended to stop after about 7–12 iterations due to condition 1.

11.1.4 Feature Pruning

The feature combination methods discussed in the next section generate enormous numbers of features. To keep the feature space tractable, features are stored in an LRU (least recently used) cache. The feature store is limited to a configurable number of elements; whenever it is full, the least recently seen feature is deleted. When a deleted feature is encountered again, it will be considered as a new feature whose weights are still at their default values.

11.2 Feature Combination Techniques

11.2.1 Sparse Binary Polynomial Hashing (SBPH)

Sparse binary polynomial hashing (SBPH) is a feature combination technique introduced by the *CRM114 Discriminator* [CRM, Yer03]. Instead of an unordered feature set, SBPH required a *ordered* list of input features. These list of input features can be the words from a tokenized text (e.g. for text classification), or the context representations described in Section 12.2, or any other suitable list of features. SBPH_N generates joined features that combine two or more input features within a context window of a specific length N (5 by default). These joined features combine the considered input features as well as their relative positions.

The idea behind this technique is the same the motivates the occasional use of bigrams and trigrams in text classification: combinations of adjacent features often transport more information than available from the mere features alone. For example, while it might be accidental if two texts contain the three features *new*, *york* and *city* (one of the might talk about New York City, the other about a new city hall in York, England), the trigram *new york city* indicates that both mention indeed the same entity. SBPH extends the idea of n-grams by also allowing "n-grams" with placeholders between features (sparse n-grams).

To generate such features, SBPH_N slides a window of length N over the list of input features. For each window position, all of the possible in-order combinations of the N input features that contain at least the newest (right-most) element of the window are

generated. For a window of length N, this generates 2^{N-1} features. Each of these joint features can be mapped to one of the odd binary numbers from 1 to $2^N - 1$ where original features at "1" positions are visible while original features at "0" positions are hidden and marked as skipped.

For the sliding window containing five input features, $f_1, \ldots, f_5$, SBPH_5 produces 16 (2^4) combined features (all containing f_5), as shown in Table 11.1.

				f_5
			f_4	f_5
		f_3	$<skip>$	f_5
		f_3	f_4	f_5
	f_2	$<skip>$	$<skip>$	f_5
	f_2	$<skip>$	f_4	f_5
	f_2	f_3	$<skip>$	f_5
	f_2	f_3	f_4	f_5
f_1	$<skip>$	$<skip>$	$<skip>$	f_5
f_1	$<skip>$	$<skip>$	f_4	f_5
f_1	$<skip>$	f_3	$<skip>$	f_5
f_1	$<skip>$	f_3	f_4	f_5
f_1	f_2	$<skip>$	$<skip>$	f_5
f_1	f_2	$<skip>$	f_4	f_5
f_1	f_2	f_3	$<skip>$	f_5
f_1	f_2	f_3	f_4	f_5

Table 11.1: SBPH_5 Feature Combinations Containing f_5

It should be noted that the features generated by SBPH are not linearly independent and that even a compact representation of the feature stream generated by SBPH may be significantly longer than the original feature list.

11.2.2 Orthogonal Sparse Bigrams (OSB)

Since the expressivity of SBPH is sufficient for many applications, we now consider if it is possible to use a smaller feature set and thereby increase speed and decrease memory requirements. For this, we consider only *pairs* of features within the window—to avoid repetitions, we require the newest member of the window to be one of the two features in the pair. The idea behind this approach is to gain speed by working only with an *orthogonal* feature set inside the window, rather that the prolific and probably redundant features generated by SBPH.

Instead of all odd numbers, only those with two bits "1" in their binary representations are used as patterns for feature generation: $2^n + 1$, for $n = 1$ to $N - 1$. With this restriction, only $N - 1$ combinations with exactly two input features are produced. We call the resulting feature combinations *orthogonal sparse bigrams (OSB)*—"sparse" because most combinations have skipped input features; only the first one is a conventional bigram. This feature combination techniques has been originally introduced by

Fidelis Assis, Shalendra Chhabra, William S. Yerazunis, and the author of this thesis in [Sie04b].

For the sliding window containing five input features, $f_1, \ldots, f_5$, OSB_5 produces four combined features (all containing f_5):

			f_4	f_5
		f_3	$<skip>$	f_5
	f_2	$<skip>$	$<skip>$	f_5
f_1	$<skip>$	$<skip>$	$<skip>$	f_5

Table 11.2 shows an example of the features generated by SBPH_5 and OSB_5 side by side. Because of the reduced number of combined features, $N-1$ in OSB_N versus 2^{N-1} in SBPH_N, classification with OSB is considerably faster than with SBPH. For the window sizes of 5 we use by default, OSB generates 4 $(5-1)$ combined features, while SBPH generates 16 (2^{5-1}) combined features for each input feature. Since both training and classification times with Winnow are linear to the number of features, training and application with OSB takes only 25% of the time required with SBPH (while still reaching comparable or even superior results, as we will see in Chap. 16).

Number		**SBPH**					**OSB**				
1	(1)					today?					
3	(11)				lucky	today?				lucky	today?
5	(101)			feel	$<skip>$	today?			feel	$<skip>$	today?
7	(111)			feel	lucky	today?					
9	(1001)		you	$<skip>$	$<skip>$	today?		you	$<skip>$	$<skip>$	today?
11	(1011)		you	$<skip>$	lucky	today?					
13	(1101)		you	feel	$<skip>$	today?					
15	(1111)		you	feel	lucky	today?					
17	(10001)	Do	$<skip>$	$<skip>$	$<skip>$	today?	Do	$<skip>$	$<skip>$	$<skip>$	today?
19	(10011)	Do	$<skip>$	$<skip>$	lucky	today?					
21	(10101)	Do	$<skip>$	feel	$<skip>$	today?					
23	(10111)	Do	$<skip>$	feel	lucky	today?					
25	(11001)	Do	you	$<skip>$	$<skip>$	today?					
27	(11011)	Do	you	$<skip>$	lucky	today?					
29	(11101)	Do	you	feel	$<skip>$	today?					
31	(11111)	Do	you	feel	lucky	today?					

Table 11.2: Features Generated by SBPH and OSB

Note that the *orthogonal sparse bigrams* form an almost complete basis set for the SBPH features—by "ORing" features in the OSB set, any feature in the SBPH feature set can be obtained, except for the unigram (f_5, the stand-alone input feature). However, there is no such redundancy in the OSB feature set; it is not possible to obtain any OSB feature by adding, ORing, or subtracting any other pairs of other OSB features; all of the OSB features are unique and not redundant.

Since the first SBPH term, the unigram f_5, cannot be obtained by ORing OSB features it seems reasonable to add it as an extra feature. However the experiments reported in Section 16.3.4 show that adding unigrams does *not* increase accuracy; in fact, it sometimes decreases accuracy.

11.3 Alternative Classification Algorithms and Implementations

As stated above, our system is agnostic to the actually used classification algorithm. Any trainable classifier can be plugged in as an alternative to Winnow, as long as a suitable interface/adapter exists and it is able to handle the large feature vectors representing the context of tokens (cf. Sec. 12.2—of course, the generated context representations are also pluggable and could be adapted for other classifiers if wished).

We decided on Winnow as the default classification algorithm because it is a fast linear-time classifier that can be trained incrementally and easily supports very large feature vectors. While Winnow is a linear separator, it is suitable for the feature combination techniques discussed above in Sec. 11.2 for enriching the feature space.

There are other Winnow implementations publicly available, but since none of them suited all our needs, we had to write our own. The *MALLET* [McC02] implementation is not used because it is neither sparse nor incremental and cannot prune features; *WEKA* [Wit99] can handle only two classes, is not sparse and cannot prune; *SNoW* [Car04] implements only simple pruning strategies (pruning features with lowest weight) and offers little support for incremental training (the prediction model is serialized and deserialized for each batch of training data).

Cohen and Singer [Coh99] use a Winnow-like multiplicative weight update algorithm called "sleeping experts" with a feature combination technique called "sparse phrases" which seems to be essentially equivalent to SBPH. Bigrams and n-grams are a classical technique; SBPH has been introduced in [Yer03] and "sparse phrases" in [Coh99]. In [Sie04b], we have introduced orthogonal sparse bigrams as a minimalistic alternative that has been new, to the best of our knowledge.

An LRU mechanism for feature set pruning has already been employed by the author in [Sie02]. We suppose that others have done the same since the idea seems to suggest itself; but currently we are not aware of such usage.

Similar to the exchangeable classifier, the usage of feature combination techniques is optional and pluggable. By default, we have used OSB_5, but it is possible to disable this or to replace it by SBPH or any other feature filter. It is also possible to chain several feature filters so one of them works on the output of the previous one.