Influence of Size on Pattern-based Sequence Classification

Menno van Zaanen MVZAANEN@UVT.NL
Tanja Gaustad T.GAUSTAD@UVT.NL
Jeanou Feijen J.H.M.FEIJEN@UVT.NL
Tilburg University, PO Box 90153, 5000LE, Tilburg, The Netherlands

Abstract

In this paper we investigate the impact of size of vocabulary, the number of classes in the classification task and the length of patterns in a pattern-based sequence classification approach. So far, the approach has been applied successfully to datasets classifying into two or four classes. We now show results on six different classification tasks ranging from five to fifty classes. In addition, classification results on three different encodings of the data, leading to different vocabulary sizes, are explored. The system in general clearly outperforms the baseline. The encodings with a larger vocabulary size outperform those with smaller vocabularies. When using fixed length patterns the results are better, but vary more compared to using a range of pattern lengths. The number of classes in the classification task does not have an effect on the trends in the results, although, as could be expected, tasks with fewer classes typically lead to higher accuracies.

1. Introduction

The area of music information retrieval (MIR) aims at extracting information from music (Downie, 2003). This includes tasks such as the automatic analysis of music, query by humming, but also the design and implementation of music archives and collections. In particular, we are interested in the sub-area of computational methods for the classification of music.

Classification of music is typically performed on datasets that contain musical pieces grouped in classes. In the most well-known tasks, pieces are grouped by genre, composer, or musical period (see e.g. (Basili et al., 2004; Backer & van Kranenburg, 2005; Kyradis, 2006; Geertzen & van Zaanen, 2008)). This comes down to designing and building a system that identifies the genre, composer, or musical period of a particular piece. Such a system allows users to search their music collection through additional views based on the meta data that is automatically assigned to the pieces.

For the classification of musical pieces, a supervised machine learning approach is commonly applied. Given a set of training data consisting of musical pieces with corresponding class information, a classification model is learned that describes how features are related to classes. The features are used to allow for a standardized and compact representation of important aspects of the musical pieces.

Features are typically divided into two groups. *Local features* describe aspects of the piece by looking at small parts of the piece, i.e. events in short time intervals (e.g. tempo changes). *Global features* encompass measures describing aspects of entire musical pieces, such as meter, distributions of intervals, or key.

In this paper, we introduce a classification system that analyzes local features from a symbolic representation of musical pieces and finds patterns that help with classification. The approach will be tested on six different classification tasks: distinguishing music by composer (two datasets), by country of origin (two datasets), by musical period and by the initial letter of the last name of the composers. Our approach is based on identifying patterns of a particular length and weighing them according to their relative importance.

The described feature extraction and classification approach is essentially task independent. It can be applied in situations where instances have an inherent sequential nature. Classification of unseen data is based on patterns found in the instances in the training data, which means that the approach relies on the ordering of the symbols within the instances.

Appearing in *Proceedings of the 20th Machine Learning conference of Belgium and The Netherlands.*

Even though the system is generic, we will illustrate the approach in the context of MIR as we can represent music in alternative ways using different encodings. Each encoding has its own properties. This allows us to investigate the relationship between said properties and the performance of the system.

The paper is structured as follows. In section 2, we describe our classification approach in more detail. The datasets that are used in the experiments, including a description of the classes and features, together with their related statistics are elaborated on in section 3. Our results are presented and discussed in section 4. Section 5 concludes the paper with a summary of our major findings and an outlook on future work.

2. Pattern-based sequence classification

The classification system (van Zaanen & Gaustad, 2010) aims at identifying patterns that can be used to classify (new, unseen) sequences. These patterns are in the shape of subsequences, i.e. consecutive symbols, and are extracted from the sequences in the training dataset. For practical purposes, patterns consist of subsequences of a pre-determined, fixed length. This means that they can be seen as n-grams, where n defines the length of the pattern (Heaps, 1978). The system only retains and uses patterns that are deemed useful according to some "usefulness" or scoring metric (described in section 2.1).

The underlying idea behind this approach is that we are aiming to identify interesting patterns for a particular class with respect to the other classes the system needs to distinguish. This is similar to an approach that learns a language model for each class, but in contrast, our approach takes the languages of the other classes into account as well. Effectively, the system learns boundaries between languages (which are represented by different classes).

By using n-grams as the representation of our patterns, we explicitly limit the class of languages we can identify. In fact, using patterns of a specific length, we can learn the boundaries of languages taken from the family of k-testable languages (Garcia & Vidal, 1990). By definition, this family contains all languages that can be described by a finite set of subsequences of length k. The k corresponds exactly with the length of the n-gram patterns. This also means that if we need to be able to identify information in sequences that span over symbols of length $> n$ (a longer distance dependency), we cannot expect the system to distinguish the correct class.

2.1. Scoring metric

During training we would like to identify patterns (i.e. n-gram subsequences) that are maximally discriminative between languages (or classes). Additionally, we would like to find patterns that occur frequently, since that increases the likelihood of the patterns occurring in the test data. Both of these properties are contained in the scoring metric that is used here.

To measure the effectiveness and usability of the patterns, we apply a classic statistical measure from the field of information retrieval, namely the "term frequency*inverse document frequency" (*tf*idf*) weight (van Rijsbergen, 1979). This measure consists of two components: term frequency (*tf*) which measures the regularity and inverse document frequency (*idf*) which measures the discriminative power of the pattern. Combined, they provide a measure that gives high weight to patterns that have a high discriminative power and, at the same time, occur frequently. Assuming that the testing data is similar to the training data, and given the fact that these patterns both occur frequently in the training data and also have a high discriminative power, this leads us to conclude that these patterns will be helpful for classification.

Originally, in the context of information retrieval, the *tf*idf* weight is used to evaluate how relevant a document in a large document collection is given a search term. In its classic application, *tf*idf* weights are computed for all documents separately in the collection with respect to a search term.

We, however, use the term "document" in a different way. Typically, in the field of information retrieval, all musical pieces would represent separate documents, but here we are interested in the usefulness of a pattern with respect to a particular class. Therefore, we take as "document" all musical pieces belonging to a particular class combined. This allows us to measure the relevance of a pattern with respect to all musical pieces that belong to the same class in one go. Effectively, we will have as many documents as there are classes in the dataset.

The first component of the *tf*idf* metric is the *tf*, which is defined as the number of occurrences of a given term in a document. Counting the number of occurrences of a term and comparing it against the counts in other documents results in a preference for longer documents (which overall contain more terms). To reduce this effect, counts are normalized by the length of the document, leading to Equation 1.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{1}$$

$n_{i,j}$ denotes the number of occurrences of term t_i in document d_j. The denominator represents the length of document d_j, which is measured as the total number of terms in document d_j.

Scoring patterns based on *tf* already provides interesting information. Assume that we are trying to identify documents that are relevant to a particular search term. If a document has many occurrences of the term, it is quite likely to be about that topic. On the other hand, if a document does not contain the term at all ($tf = 0$) then that document is probably not about this term. Obviously, the document may still be about the same topic, but might use different words, such as synonyms, for the topic. In fact, this has led to research into, for example, lemmatizing, stemming, pseudo relevance feedback and automatic synonym generation (Baeza-Yates & Ribeiro-Neto, 1999).

Another problem with simply using the *tf* scoring metric is that there are terms that occur very frequently in all or at least most documents. For instance, function words, such as *the*, *a*, and *and*, probably occur in all documents. When a function word is used as a search term, documents with many function words are considered very relevant (since the *tf* will be very high). However, the weight given to function words will also be higher than that of content words.

To reduce the effect of the function words, or often occurring terms that appear in all documents, the *tf* metric is extended with an *idf* component as described in Equation 2.

$$idf_i = \log \frac{|D|}{|\{d : t_i \in d\}|} \tag{2}$$

$|D|$ is the total number of documents in the collection and $|\{d : t_i \in d\}|$ is the number of documents that contain the term t_i.

The *idf* metric measures relevance of a term with respect to all documents. In other words, terms that occur in all documents are less useful when deciding document relevance compared against terms that occur in fewer documents. When relevance is computed based on a set of terms the *idf* gives higher weight to terms that only occur in fewer documents.

To obtain the *tf*idf* weight for a particular term, the term frequency *tf* and inverse document frequency *idf* are combined as shown in Equation 3.

$$tf*idf_{i,j} = tf_{i,j} \times idf_i \tag{3}$$

The default way of computing *tf*idf* provides us with an indication of how relevant a particular document is to a particular term. This metric can be extended to multiple search terms. Once the *tf*idf* for all documents is computed for each term, the *tf*idf* values are summed and the most relevant document, i.e. with the highest *tf*idf* score, is selected.

Here, we extend the *tf*idf* metric differently. In addition to combining the *tf*idf* values of a set of terms, we compute *tf*idf* values of sequences of terms thereby preserving the underlying order. This amounts to using n-grams (with $n \geq 1$) as terms. Our intuition is that subsequences are more informative than single terms to determine boundaries between languages.

The modification of the computation of the *tf*idf* weights is rather straightforward. Instead of counting single terms, n-grams are counted as if they are single terms (with single terms being a specific case where $n = 1$). For instance, $n_{i,j}$ is the number of occurrences of a particular n-gram t_i in document d_j.

2.2. System walk-through

During training, the system receives a collection of classes, each consisting of a set of sequences that come from the underlying language of that class. Based on these sequences, all possible n-grams are extracted and for each of these patterns the *tf*idf* score is calculated. This leads to a set of patterns combined with a score for each class. A score signifies how well the pattern fits the particular class (the *tf* of the class) and is weighted by the *idf* (which indicates uniqueness of the pattern with respect to the other classes). Patterns that have a *tf*idf* score of zero are removed, as they are useless for classification purposes. This may happen when the pattern occurs in all classes.

After the training phase, the system receives a new sequence which needs to be classified. Based on the sequence, the system computes an overall score for each class. This is done by applying each pattern to the sequence and for each match, the *tf*idf* scores for that pattern are added to the cumulative scores for each class. Finally, the system selects the class that has the highest score as the "correct" class for the sequence.

We have described a system that uses fixed length patterns, but it is also possible to use patterns of varying length. This is arranged by computing scores of a sequence using patterns of different length. The final score of the sequence is the linear combination of the scores of each pattern length.

Taking another look at using long and short patterns simultaneously, we see that shorter patterns typically have a higher *tf*idf* score. This is due to the fact that shorter patterns have a higher overall likelihood of occurring in a sequence than longer patterns. Since the

number of occurrences is taken into account (in the *tf* component of the *tf*idf* score), this means that shorter patterns typically have a higher score than longer patterns. On the other hand, if a long pattern, which is more precise, is found in a sequence during classification, it should probably have more influence in the overall decision. We have experimented with (multiplicative) length normalization of the patterns, but this did not have any significant effect.

3. Data

3.1. Dataset and classification tasks

To analyze and evaluate our approach, we retrieved data that can be found under the "composers" section on the front page of the **kern scores website[1] (Sapp, 2005). This data contains symbolic representations of musical pieces from 50 composers. Using these pieces, we compiled six different classification tasks. All tasks use pieces from this one dataset. Table 1 provides an overview of the six tasks including the number of classes and number of musical pieces.

The first two tasks involve classifying the musical pieces according to their composer. The first dataset consists of the pieces of all 50 composers (*composers*). However, we found that there are several classes (composers) that only have a small number of pieces in the dataset. To remove some of the skewedness, we created a second dataset by imposing a frequency threshold and retaining only those composers that have 50 or more musical pieces in the original set (*composers50*).

The next two tasks aim at classifying the pieces according to the country of origin of its composer. Again, two datasets are created, on the one hand using all data (*countries*) and on the other hand, similarly to the division in the composers tasks, restricting the classes to countries with at least 50 pieces (*countries50*).

The fifth task involves classification into musical periods (*periods*). Each piece is assigned to one of six periods: Middle Ages (400–1400), Renaissance (1400–1600), Baroque (1600–1750), Classical (1750–1800), Romantic (1800–1900), Modern (1900–today). The information required for assigning countries and musical periods of composers was taken from the International Music Score Library Project (IMSLP) website[2].

The final task has been setup as a sanity check: The classes contain pieces of composers that share the first letter of the last name (*initials*). Since the system aims at learning regularities within (and between) classes and since we do not assume any real regularities within pieces of these artificial classes, we expect the performance here to be lower than that of other tasks.

[1] `http://kern.ccarh.org/`

[2] `http://www.imslp.org/`

Table 1. Overview of the six classification tasks.

Dataset	# classes	# pieces
composers	50	3132
composers50	9	2747
countries	14	3132
countries50	5	3050
periods	6	3132
initials	17	3132

3.2. Data representation

Starting with a set of musical pieces in **kern format (Huron, 1997), the music is transformed into different symbolic representations. The **kern format is an ASCII representation of sheet music. Notes (and other musical symbols) are grouped by voice, which allows us to extract notes of a particular voice. The experiments described here are all use the notes contained in the first voice in the **kern files. By applying a transformation on the notes, we can select which properties of the data we would like to use while working with the same underlying data.

There are two musical aspects we extract information from: pitch and duration. For both aspects, three different encodings are used: *absolute*, *relative*, and *contour*. In this paper, the same encoding is used for both pitch and duration. Note that many more encodings and combinations of encodings are possible, but since that will explode the experimental space we only focus on these three encodings here.

The *absolute* encoding for pitch refers to the absolute value in semitones (e.g. $c = 0$, $d = 2$, $e = 4$, $c' = 12$, etc.). For duration a similar encoding is used where each note length has its own symbol. The *relative* encoding describes pitch changes with respect to the previous note in the number of semitones and direction of change (e.g. 4, -7). A similar encoding is designed for duration, which describes the relative duration of consecutive notes. Finally, *contour* only models whether the pitch or duration rises (1), falls (-1) or stays the same (0). Contour encoding is also known as the Parsons Code (for Melodic Contours) (Parsons, 1975).

The major difference between the various encodings is that the number of unique symbols (i.e. the size of the vocabulary or the number of types) is significantly different. The classification tasks do not have any influence on the number of unique symbols. Given

Table 2. Mean number of unique symbols per encoding and their standard deviation.

Encoding	Mean # of unique symbols
Relative	1075 (±75.69)
Absolute	727 (±57.75)
Contour	12 (±00.00)

Table 3. Mean number of useful patterns per encoding and pattern length.

	Pattern length			
	1	2	3	4
Relative	2005	25275	83291	147305
Absolute	1047	27018	108467	189957
Contour	4	111	1668	12079

the same encoding, the size of the vocabulary will remain constant across different tasks. An overview of the mean number of unique symbols per encoding and their standard deviations can be found in Table 2.

Another difference between the encodings is the amount of information described by the symbols. The contour encoding only contains information on the direction, whereas the relative encoding describes the interval. However, it does not have information on the exact notes being used. The absolute encoding denotes exact notes, so all information is encoded. However, it may be argued that the relative encoding fits better with how patterns in music are identified. Sequences of notes are recognized as similar even when the sequence starts on different notes.

4. Results

When applying[3] the system to the datasets based on the different encodings there are many aspects that can be investigated. Firstly, we will look at the number of patterns that are generated. In section 3.2, we already saw that the classification task does not have a significant impact on the number of unique symbols. Also, the classification task does not have a significant impact on the number of patterns, so we only show this information per encoding and n-gram size. As can be seen in Table 3, for all encodings the number of useful patterns (based on the *tf*idf*) grows when the patterns get longer. This is due to the fact that longer patterns have a lower likelihood of occurring in all classes (which would mean that the *idf* is zero). Furthermore, the contour encoding finds fewer useful patterns. This is because the number of unique symbols is much smaller (see Table 2), which means that it is more likely that the patterns occur in all classes.

Comparing the changes in number of patterns between relative and absolute encoding, the following picture emerges. At first, larger vocabulary sizes (relative encoding) lead to more possible patterns as can be seen when the pattern is of length one. However, when the patterns get longer, the absolute encoding leads to more patterns. This is because the number of (combinations of) intervals per class is more likely to occur within different classes than the exact combination of notes. In other words, the relative encoding, which describes intervals, generalizes over the absolute encoding, which describes exact notes.

Next, we look at the accuracy results for experiments where all patterns have the same length (fixed length n-grams). These results are depicted in Figure 1.

The first thing to notice is that the systems overall significantly outperform the majority class baseline (although in some cases the accuracies are lower than the baseline). In general, longer patterns perform worse, which is expected as long patterns are not likely to reoccur in the test data. Similarly, very short patterns (n=1) do not perform well, as the patterns that are retained (often accidentally) do not occur in all classes. In this case, the short patterns give a false sense of predictive power. If no patterns match the test sequence, the system falls back on the majority class baseline.

Considering the different encodings, it is clear that the contour representation performs worst. However, results get better with longer n-grams. This can be explained by the more slowly growing pattern space (see Table 3). The absolute and relative encodings perform similarly (there is no statistically significant difference between absolute and relative encodings, but the contour encoding performs significantly worse) and both peak at $n = 3$ or 4 depending on the classification task.

Comparing the classification tasks with and without frequency cutoff (*composers* vs. *composers50* and *countries* vs. *countries50*), we can observe an increase in accuracy for the two tasks with a frequency cutoff. The most probable explanation is the reduction of classes involved. Keep in mind that the classes that have a small number of pieces are removed in the frequency cutoff datasets.

Interestingly enough, our classification system produces results that lie above the majority class baseline for the *initials* task albeit the accuracy is overall

[3] All results presented in this paper are computed using 10-fold cross validation.

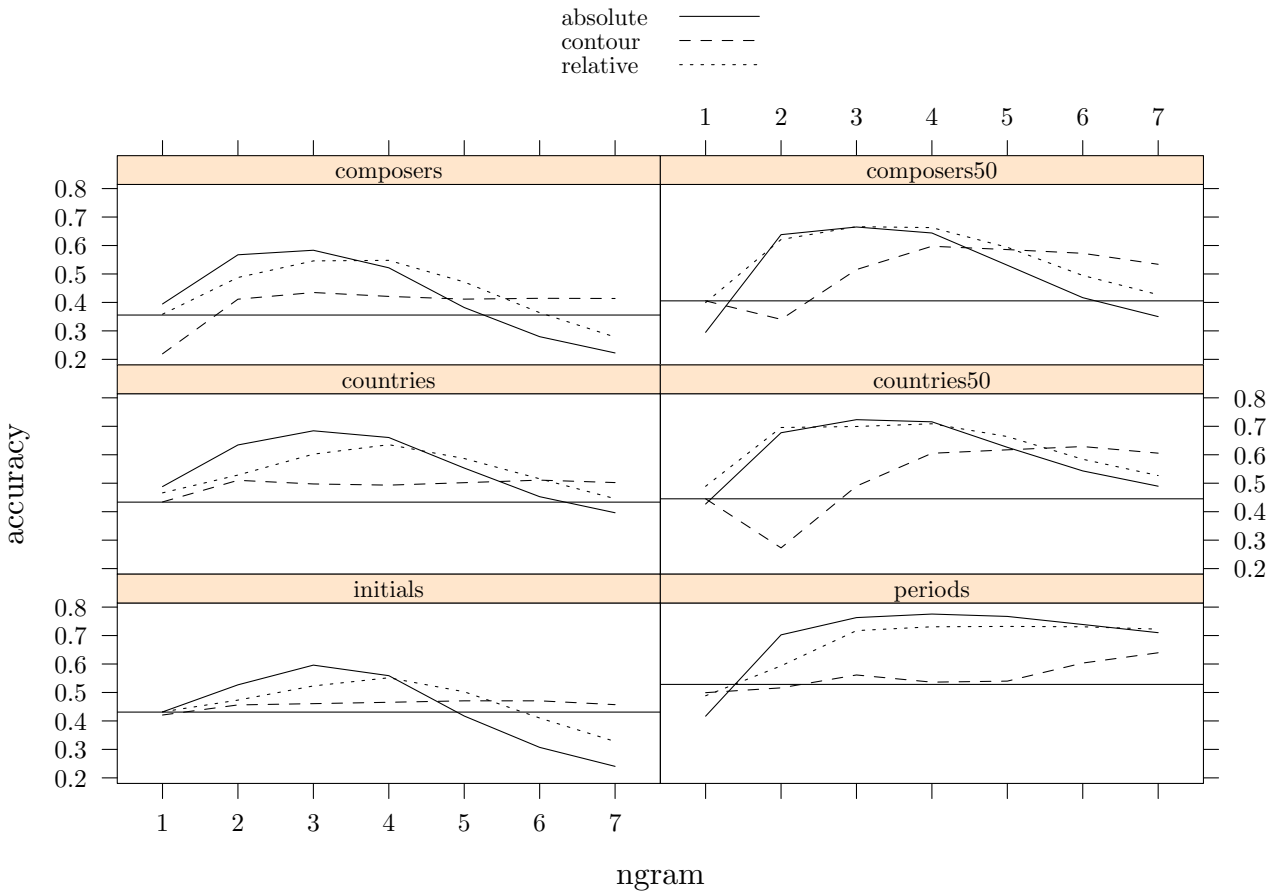

Figure 1. Accuracy of patterns of n-grams of one length. The horizontal line is the baseline.

the lowest. On inspecting the datasets, it turns out that there are strong relations between the different classification tasks. When cross-tabulating the *initials* task with the *countries* task, we can see that if for each of the countries classes (14 classes) we pick the most frequently co-occurring initials class (out of 17 classes), 71.8% of the data is described. Also, cross-tabulating the periods task (6 classes) and again taking the initials class (out of 17 classes) with the highest co-occurrence count, 55.3% of the pieces are explained.

Taking this into the extreme, we can also examine the composers task. Assume that we have a perfect classifier for this task. We can then learn a direct mapping from the class of composers to that of the initials. Obviously, this is possible, since there exists a simple relationship between the composer classes and the initials classes. In other words, if we know the composer, we can select the correct initials class perfectly.

However, during the initials task, the system does not have complete knowledge on the classes of the composer task. If we reverse the co-occurrence in the cross-tabulation, we see that when classifying into composers (50 classes) from the initials task (17 classes) it is still possible to classify 81.1% correctly. Similarly, we can correctly classify into the countries dataset (14 classes) 89.6% and into the period dataset (6 classes) 83.5% of the data. Overall, there is a large overlap between the classes from the different classification tasks.

Finally, we investigate the accuracy results when we allow patterns to have different lengths (specified within a range) in the classifier. The results of these experiments can be found in Figure 2.

Comparing the results (to fixed length n-grams in Figure 1), the curves are generally flatter and the results somewhat lower. We still see the expected increase in performance with a frequency threshold/reduction in classes, but no more peaks are evident. Again, the results for relative and absolute encodings are comparable, whereas contour performs significantly worse.

The reason for these flatter curves is that, as we can see in Figure 1, the longer n-grams occur less frequently (hence the lower accuracy), but in that case the shorter

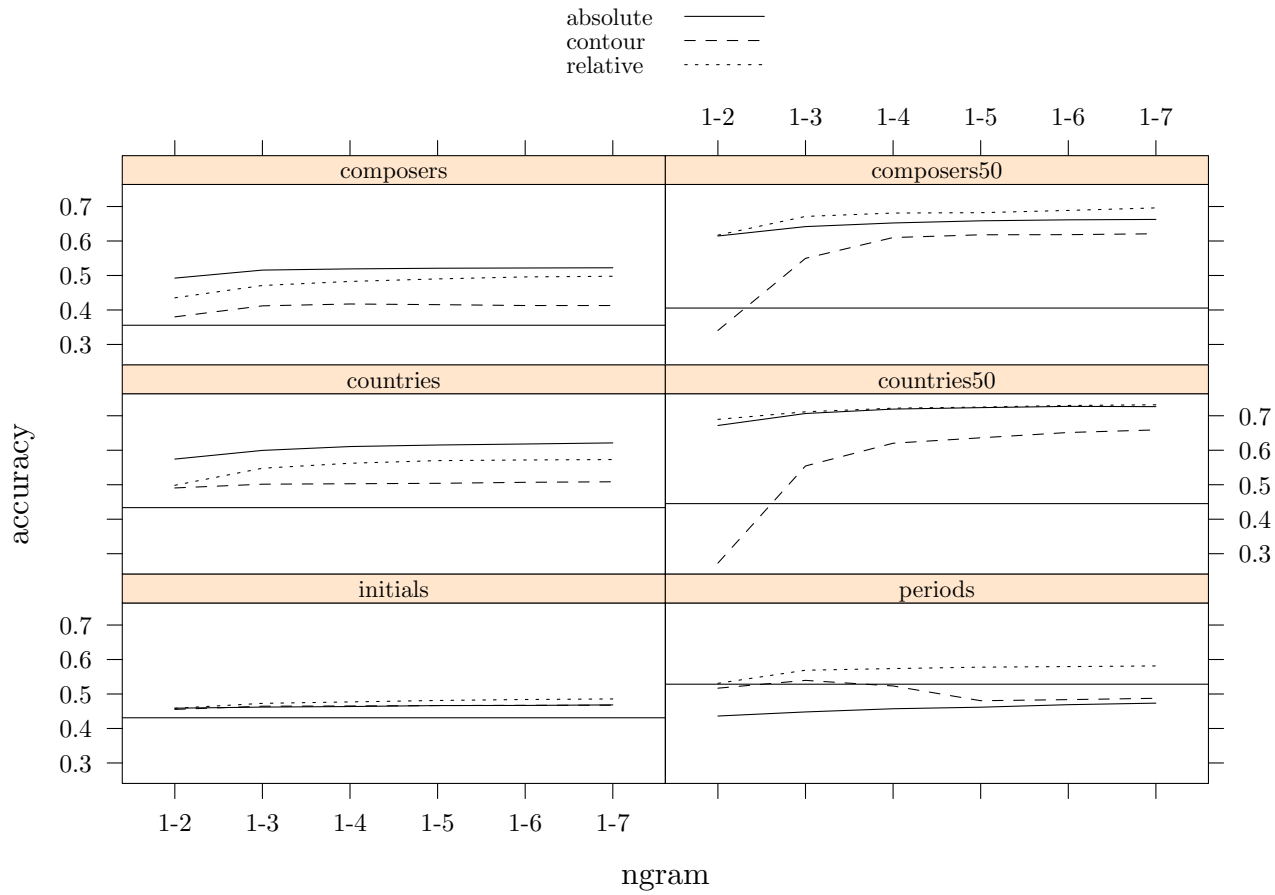

Figure 2. Accuracy of patterns of n-grams of a range of lengths. The horizontal line is the baseline.

n-grams take over. Furthermore, smaller n-grams may actually be contained in the longer n-grams, which then means that they are effectively used twice.

The encodings lead to similar results as in the single n-gram case: the difference between absolute and relative encodings are not statistically significantly different, whereas the contour encoding generally performs worse than the other encodings.

The range results also show some aspects that are harder to explain. Consider, for instance, the periods task where the absolute encoding (and the contour encoding with larger n-grams) falls below the baseline. We currently do not have an explanation for this.

For the task of classifying according to the first letter of the last name of a composer (the *initials* tasks), we now see the expected outcome: a flat curve (just above the baseline) which indicates that the patterns have only minimal influence on classification.

Comparing the use of fixed length n-grams and range-based n-grams, the results show that using n-gram ranges leads to more robust results. Taking the datasets and encodings into account, there is a statistically significant difference between the single or range n-grams.

5. Conclusions and future work

In this paper we have described a classification system that aims at finding regularly recurring patterns that have a high degree of discrimination among different classes. The well-known *tf*idf* metric is used to find and score the patterns. This score is also used to identify the best matching class for new, unseen sequences the system tries to classify.

To illustrate different aspects of the system, we have applied it to one dataset, but in different classification tasks. In particular, taking a dataset containing musical pieces, we provided classification tasks based on names of composers, the countries the composers come from, the musical periods the composers belong to and the initial letter of the last name of the composer.

We also described three different encodings of the music, namely absolute (absolute values of pitch and duration), relative (taking a previous note into account) and contour (direction of pitch and duration).

Overall, the system performs significantly better than the majority baseline. There is no difference between the absolute and relative encodings and both outperform the contour information (which encodes less information and hence has a much smaller vocabulary).

When allowing the system to use patterns of one fixed length only, the results are better, but the choice of the pattern length is important. Patterns that are too short or too long reduce the accuracy of the system. This is no problem when a range of lengths of patterns is allowed as the results are more robust. However, the overall results are lower.

The use of multiple classification tasks on the same dataset has nice properties, as it keeps the amount of data in each classification task constant. This allows us to experiment with different numbers of classes (per classification task). However, it also gives rise to problems. For instance, there is a direct relationship between the different tasks, which makes it difficult to identify exactly what the system is learning. This is illustrated in the initials dataset, which we intended as a sanity check, but instead performed reasonably well, as it (partly) coincided with other classification tasks.

In future work, we would like to experiment on a range of datasets, perhaps also outside the musical domain. The aim of using different datasets is to get a better grip on when exactly the system breaks down, in particular with respect to the size of the vocabulary.

Furthermore, we plan to explore different scoring metrics. At the moment, we have used one *tf*idf* metric, but alternatives exist (Manning et al., 2008, p. 118). This is particularly interesting because we use the *idf* component in the scoring metric in a very small document collection (since each class is considered a document). The choice for this component may have a large impact on the overall performance of the system.

References

Backer, E., & van Kranenburg, P. (2005). On musical stylometry: a pattern recognition approach. *Pattern Recognition Letters*, *26*, 299–309.

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern information retrieval*. Reading:MA, USA: Addison-Wesley Publishing Company.

Basili, R., Serafini, A., & Stellato, A. (2004). Classification of musical genre: a machine learning approach. *Proceedings of the fifth International Conference on Music Information Retrieval (ISMIR); Barcelona, Spain*.

Downie, J. S. (2003). Music information retrieval. *Annual Review of Information Science and Technology*, *37*, 295–340.

Garcia, P., & Vidal, E. (1990). Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *12*, 920–925.

Geertzen, J., & van Zaanen, M. (2008). Composer classification using grammatical inference. *Proceedings of the MML 2008 International Workshop on Machine Learning and Music held in conjunction with ICML/COLT/UAI 2008, Helsinki, Finland* (pp. 17–18).

Heaps, H. S. (1978). *Information retrieval: Theoretical and computational aspects*. New York, United States: Academic press.

Huron, D. (1997). Humdrum and kern: selective feature encoding. In E. Selfridge-Field (Ed.), *Beyond MIDI: The handbook of musical codes*, 375–401. Cambridge:MA, USA and London, UK: Massachusetts Institute of Technology Press.

Kyradis, I. (2006). Symbolic music genre classification based on note pitch and duration. *Proceedings of the 10th East European Conference on Advances in Databases and Information Systems; Thessaloniki, Greece* (pp. 329–338).

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to infomation retrieval*. New York, NY, USA: Cambridge University Press.

Parsons, D. (1975). *The directory of tunes and musical themes*. Spencer-Brown.

Sapp, C. S. (2005). Online database of scores in the humdrum file format. *Proceedings of the sixth International Conference on Music Information Retrieval (ISMIR); London, United Kingdom* (pp. 664–665).

van Rijsbergen, C. J. (1979). *Information retrieval*. Glasgow, UK: University of Glasgow. 2nd edition, Printout.

van Zaanen, M., & Gaustad, T. (2010). Grammatical inference as class discrimination. *Grammatical Inference: Theoretical Results and Applications; Valencia, Spain* (pp. 245–257). Berlin Heidelberg, Germany: Springer-Verlag.