A Quick Sequential Forward Floating Feature Selection Algorithm for Emotion Detection from Speech

Mátyás Brendel, Riccardo Zaccarelli, Laurence Devillers

LIMSI-CNRS, France
{mbrendel,riccardo,devil}@limsi.fr

Abstract

In this paper we present an improved Sequential Forward Floating Search algorithm. Subsequently, extensive tests are carried out on a selection of French emotional language resources well suited for a first impression on general applicability. A detailed analysis is presented to test the various modifications suggested one-by-one. Our conclusion is that the modification in the forward step result in a considerable improvement in speed (~80%) while no considerable and systematic loss in quality is experienced. The modifications in the backward step seem to have only significance when a higher number of features is achieved. The final clarification of this issue remains the task of future work. As a result we may suggest a quick feature selection algorithm, which is practically more suitable for the state of the art, larger corpora and wider feature-banks. Our quick SFSS is general: it can also be used in any other field of application.

Index Terms: emotion detection, feature selection heuristics

1. Introduction

Feature selection methods are divided in two big groups: feature ranking and wrapper methods. A good introduction can be found in [1]. Feature or variable ranking is usually computationally affordable, since only a simple scoring function is computed. However, it usually cannot take into account the interaction between features: “a variable that is completely useless by itself can provide a significant performance improvement when taken with others”. Wrappers utilize the learning method as a black box, consequently they are time-consuming.

Sequential Floating Forward Selection (SFFS) was introduced in [2] and is certainly still among the most widely used techniques in the field. It is worth however to mention that at the time when dSFFS was developed, available corpora were smaller and there has also been a huge expansion in the number of usable features since then.

SFFS is the most important feature selection method used in emotion detection from speech. Nevertheless, as it is illustrated for example in [3] improvement in speed is important in this field of application, even with the risk of decreasing performance. The paper presents a method for feature selection which is similar to racing used in parameter tuning: when multiple cross-validations are carried out as a test the presented method estimates the result with fewer tests through a statistical probe. Note that they do not modify the search strategy of SFFS itself. The authors of [3] claim that the proposed method speeds up SFFS by 60%. We will aim to a higher speed up (~80%). Moreover, our improvement is in the search strategy itself, thus it may be combined for example with the method presented in [3]. Such improvement in speed is crucial even if parallel computing is considered, since on the other hand feature-banks with thousands of features are available.

In section 2 we present the corpora, features and evaluation method we use in this paper, in section 3 we present our quick SFFS method; in section 4 we present the first, proof of concept test. In section 5 we give a more detailed analysis of the components of our modification. In section 6 we summarize our conclusions and sketch up the possible future work.

2. Corpora

The corpora CINEMO and JEMO as well as the feature-banks were already introduced in [4]. Here we only give a short description since our main focus is on feature selection.

The corpus CINEMO [5] used in this paper consists of a subset of 1532 instances after segmentation of emotional French speech amounting to a total net playtime of 2:13:59 hours. 50 speakers (of 15 to 60 years old) dubbed 27 scenes of 12 movies. A subset of the more consensual segments was chosen for training models for detection of 4 classes (POSitive, SADness, ANGer and NEUtral). JEMO is a corpus collected with an emotion-detection game built on CINEMO data. The corpus JEMO features 1135 instances after segmentation of speech recorded from 39 speakers (of 18 to 60 years old). The data were more prototypical than in the corpus CINEMO because very few mixtures of emotions were annotated.

In the following we will describe two feature-sets. First the LIMSI features used in this paper. Each speech segment is passed through spectral (16 MFCCs) and prosodic analysis (pitch, zero-crossing and energy) by the LIMSI extractor. The feature extractor next calculates basic statistical features on voiced parts: min, max, mean, standard deviation, range, median quartile, third quartile, min and max intra range and the mean and standard deviation of the coefficients of least square fitting regression (of each voiced segment); min and max inter range (between voiced segments). Overall, 458 features are thus obtained including further post-processing: 23 for pitch, 51 for energy (from these 22 root mean square energy), 18 zero-crossings and 366 for MFCC1–16.

To introduce sufficient variance in our experimentation and not base our findings solely on one feature extractor, we use the openEAR toolkit’s “base” set with 988 features based on 19 functionals of 26 acoustic low-level descriptors (LLD, smoothed by simple moving average) and corresponding first order delta regression coefficients.
3. Feature selection with set-similarity based heuristic

Let $Y = \{y_i : 1 \leq i \leq D\}$ denote a set of available features and $X = \{x_i : 1 \leq i \leq k, x_i \in Y\}$ a subset of $Y$. The so-called wrapper methods carry out feature selection by running a real test on the subset $X$.

The classical SFFS can be considered as a kind of "tree-search", where the nodes represent possible feature-sets and the edges possible transition-steps. We have to note that the graph is actually not a tree, since one can reach a node in many different ways and a child has several parents. Actually, this graph may be called the Hasse-diagram of the feature set’s power set partially ordered by inclusion. Each edge represents a possible transition between a parent and a child. The transition may be in two directions.

The sequential forward selection (SFS) is a parent-to-child transition: a full scan of all the children of the actual parent node is done and the best one is taken. Note that this is already a pruning and a heuristics, which may be sub-optimal. There is also a backward step (SBS): a child to parent transition, where a feature is removed. If the resulting parent is better than the best visited node so far at level k, the update is made along the transition. In the classical SFFS both SFS and SBS carry out a full scan of all the possible nodes.

Feature selection can also be considered as a global optimization task. In this context the forward step can be considered as more of an exploitation and the backward step as more of an exploration, but things are more complicated as we will see later. In global optimization the balance of exploration and exploitation is important: without the backward step, SFFS would be too greedy: it would more likely stuck in local optimum. However, what we will show in our paper is that the backward step is strong enough to make the forward step more greedy.

At a certain level $k$, the classical SFS tests all the $D - k$ nodes and takes the best. A more greedy algorithm could take the first child with a positive gain. This way exploitation is made faster, which however, at a cost of increases the danger to stick in a local optimum. In other words, a greedy SFS sacrifices exploration for speed. Nevertheless, our claim is that in our field of application, the backward step contains sufficient exploration to handle this.

A further improvement can be made to order the new feature-candidates according to the expectation of their significance. Note that ordering the child nodes makes only a difference of the scan is not full. According, ordering in classical SFFS makes no sense, however it is important for a greedy SFS: we would like to visit first the better candidates.

Let us assume that we are at level $k$ at a node with feature set $X$. We will call the gain for feature $x$ achievable by the parent to child transition, i.e. by adding feature $x$ to the set of features $X$ as the significance of $x$: $S(x, X)$. To estimate $S(x, X)$ we use the known history of our search tree: we take the most similar case, when $x$ was added to a feature set $X'$. The estimated significance is denoted as $S'(x, X)$.

Similarity of two sets may be measured in many ways. We have chosen one of the most used: the Jaccard similarity. This means that estimated significance is computed as follows:

$$S'(x, X) = J(X^* \cup \{x\}) - J(X^*) :$$

$$X^* = \arg \max_X Jaccard(X, X') : \frac{|X \cap X'|}{|X \cup X'|}$$

where Jaccard() is the Jaccard set similarity measure and $J$ is the evaluation value for a given feature set. Instead of "argmax", other function may be used, like for example a weighted sum with exponentially decaying weights. Children are ordered in decreasing order of estimated significance to start with the most promising ones.

In the backward step of SFFS one can not afford to be very greedy, since a strong exploration is needed to avoid local minima. However, we may avoid a full scan, and carry out only a partial scan: only a certain percent of the parents are to be tested. In this case parents are ordered in increasing order of estimated significance, $S'$, so that we first try to remove the least significant features. We applied a $p = 20\%$ scan of all the parents, which proved to be enough in our field of application.

We call our method as "SFFS with Set-Similarity Heuristics" (SFFS-SSH). Our algorithm has only one parameter to be tuned, that is $p$. Note that the computational overhead of our heuristics is negligible compared to a 10-fold CV test on a corpus. Thus the running time of the algorithm depends mainly on the number of iterations, which we expect to decrease significantly with our modifications.

4. Proof of concept

We trained the data set using LIBSVM [7] with a radial basis function kernel. We use 10-fold speaker independent cross-validation, designed in our lab [8]. This means in short that speakers were divided into folds, instead of instances, thus maintaining speaker independence, while doing a10-fold CV. Note that this covers the full corpus. We omitted a separate validation set, since we are interested to compare two methods and not validate the final performance.

<table>
<thead>
<tr>
<th># it. and RR</th>
<th>SFFS (24)</th>
<th>SFFS-SSH (24)</th>
<th>all feat.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIMSI (458)</td>
<td>28382/53.4%</td>
<td>59.6%</td>
<td>54.6%</td>
</tr>
<tr>
<td>Openear (988)</td>
<td>28126/58.5%</td>
<td>639454.2%</td>
<td>474258.8%</td>
</tr>
</tbody>
</table>

Table 1: Number of iterations and Recognition Rate of the best 24 selected features on the united corpus (CINEMO+JEMO) with classical SFFS and our modification (SFFS-SSH). Speed-up computed in number of iterations is $77\%$ and $83\%$ respectively.

Table 1 shows the results with a fixed number of 24 features. As it can be seen, selecting 24 features does not improve the result. We will see better results with our algorithm later. It also can be seen that SFFS-SSH provides similar good feature-selection, but takes much shorter time. Thus the first confirmation of our method was successful.

5. Detailed tests and analysis

We suggested several modifications to the classical SFFS algorithm which were planed to work together. Therefore several variants of the algorithm were tested to see the change of performance and to analyze the contribution of each component of the modification.

5.1. Forward Step

First we tested the SFS part of the algorithm omitting the SBS. Note that all subsequent tests are made with the openEAR features, since they being of a higher number – demonstrate better the benefits of our algorithm and we also had better results with them.
tant effect is that the features are better. The algorithm SSHSFS same 8000 iterations more features are added, the more important helps to select better features. A minor effect is that in the RR-NF diagram.

It is not easy to illustrate and analyze the progress of the algorithms, we will use two types of figures, the first one depicts recognition rate in function of the iterations (RR-NI diagram), the second recognition rate in function of the number of features (RR-NF diagram).

Figure 1 shows the RR-NI diagram of three algorithmic variations using only the forward step. One point represents one evaluation of the criterion function. The figure contains only the first 8000 iterations, but this is sufficient to demonstrate the benefits of SSHSFS.

The algorithmic variations tested here are as follows: (1) SSHSFS, i.e. our proposed forward step, (2) GreedySFS, i.e. a greedy forward step without heuristics, and (3) classical SFS, i.e. the classical forward step. Remember that no backward step is used in these cases. We first want to clarify the possibilities of the forward step alone.

Note that each algorithm starts with the evaluation of all the features one-by-one. Since we have 988 features, the first 988 iterations of each algorithm is the same. After this first cycle however, the variations differ dramatically: the classical algorithm progresses only very slowly. The greedy algorithm starts up with a much higher pace. We can also see that our SSH adds an additional improvement even to the greedy algorithm. This means that the heuristics is valid.

The RR-NI diagram is suitable to show how fast exploitation is made. What is missing however, is that we have no information about the number of features. Improvement in RR may be due to good features, but it may also result from more features.

The RR-NF diagram is perfect to test this: it shows how good is the exploration in our algorithm. Figure 2 shows the RR-NF diagram.

As it can be seen SSH is better than greedy SFS, since it really helps to select better features. A minor effect is that in the same 8000 iterations more features are added, the more important effect is that the features are better. The algorithm SSHSFS progresses on a higher path than the GreedySFS algorithm. In the RR-NF diagram the important detail to observe is always the upper, limiting curve of the set of points. This indicates, how thorough exploration is achieved. Our modifications aim to reasonably sacrifice some exploration in the name of speed. This can be achieved at the cost of very minor and not even consistent differences, as can be seen by analyzing the RR-NI and RR-NF diagrams.

It is also obvious from figure 2 that we have better results here because there are much more features as at the results in table 1. Unfortunately we had not the time to run classical SFFS until much more features because of its enormous demand in processing time, consequently we are not able to present a comparison with significantly more features.

Figure 3 shows how quick the possible forward steps add the new features as the iterations progress. Classical SFS tries all the possible features in one SFS cycle. That is why the classical SFS progresses very slowly: it takes almost 1000 tests to add one new feature. In general, if we have D features, classical SFS will carry out \( D - k \) test on level \( k \). This means that if we would like to get \( d \) features, \( dD - d^2/2 \) iterations are needed. On the other hand for our SSHSFS this is only the worst case, practically \( c \) times \( d \) number of iterations are needed, where \( c \) is the average number to find the first feature with gain. Most likely \( c \ll (2D - d)/2 \), especially for small \( d \).

The algorithm SSHSFS starts up to improve very quickly in figure 1, but then it slows down quite early. In figure 2 however, it can be seen that this slowing down happens at a high number of features. One can conclude from this that after thousands of iterations even SSHSFS spends most of its time to find features with an actual gain. This means that the early flattening of the curve in figure 1 is more due to the fact that finding good features to add becomes difficult. The real saturation, when none of the features improve the performance is only reached at the end, around 140 features.

5.2. Backward Step

After testing various forward steps, we also tested variations for SBS. The algorithmic variations tested are as follows: (1) Classical SFFS, (2) SSHSFS with classical SBS, i.e. our SFS with full scan in SBS, and (3) SFS-SH, i.e. all of our modifications.

Figure 4 shows the RR-NI diagram for these tests. It is obvious that classical SFFS progresses tremendously slowly. SFFS-SSH seems to progress very quickly at the beginning, but it reaches a local optimum early and it is not easy for it to improve further, since exploration is weaker. SSHSFS+ClassicalSBS starts slower – but considerably quicker than classical SFFS – however, this seems to pay back in the end.
In the RR-NF figure in figure 5 our first observations are confirmed. One can see that classical SFFS progresses very slowly but with a very thorough exploration: classical SFFS reaches the highest curve, unfortunately it takes a lot of time and the benefit is very minor compared to the time-cost and disappears at 24 features (see table 1). The algorithm with the second strongest exploration is SSHSFS+ClassicalSBS. This is fast enough and it also achieves a very high curve. It seems to be practically the best choice, a good compromise between speed and quality.

One has to realize that the full scan in the classical SBS at level k takes only k tests. Since k is small at the beginning we can afford to carry out a full backward scan. This is the case until k gets higher. Until we are far from D, the full number of features, our modification of SBS is not really important. It would however be needed as k gets higher.

Table 2 shows the results of various algorithms. Our SFFS-SSH achieves a good result with a reasonable number of features. As we already know, our forward step performs slightly better with the classical backward step, but that is not the case with fewer number of iterations and might not be the case with D or d higher. We also tested a different values for p: 50 percent of features were tested in SBS. This performed slightly lower. Considering that ClassicalSBS is the same as p=100, we can conclude that the value of p makes slight differences in a reasonable range and it is not easy to optimize it, since it does not change monotonically the result. Parameter tuning has to do the work.

### 6. Conclusions and future work

In global optimization algorithm speed and quality of the result is usually mutually exclusive: it is not easy to improve an algorithm in both aspects. In the two dimensional space of speed and quality of the result there exist the so called Pareto-front, where one dimension can only improve at the cost of the other. Classical SFFS is probably close to the Pareto-boundary, where speed and quality of the result is acceptable for few hundred of features. As the number of features increases to several hundreds or thousands, however, the speed of classical SFFS is not suitable. Since this is the situation at current corpora and feature-banks, further improvement of speed is needed.

Our fast SFFS algorithm is a practical compromise for feature selection tasks in the future, where the number of features and the size of the corpus is considerably large. Our algorithm lies in a region where speed is improved significantly with negligible cost in quality. With the help of the set-similarity heuristics even some improvement in quality is possible at least in some cases. Thus we are probably getting closer to the Pareto-front with our heuristic.

Our SFFS-SSH can be combined with other improvements like the method described in [3]. This way we took a big step forward to ensure that wrapper method feature selection can practically be used even for larger corpora and feature banks.

Future work shall compare classical SFFS and our SFFS-SSH by considerably longer runs achieving considerably more features with classical SFFS.

We also indicated possibilities to develop our heuristics to obtain a more complex, possibly better estimation of gain related to adding a feature.

### 7. References


