CS229 Project Report
Polyphonic Piano Transcription

Mohammad Sadegh Ebrahimi SADEGH@STANFORD.EDU
Stanford University

Jean-Baptiste Boin JBBOIN@STANFORD.EDU
Stanford University

1. Introduction

In this project we want to employ machine learning algorithms to extract the notes that are played in a polyphonic piano song. There has been a lot of research on music transcription recently, but most of them are aimed at monophonic identification. In this project, we looked at the problem in a more general way and tried to improve the performance using different techniques(1). One of the significant differences between using a monophonic and polyphonic song is that in polyphonic identification we cannot use the information that at most one note is playing, so techniques using multiclass classifiers are not applicable. Depending of how we apply our algorithm, we found that there was a trade-off between sensitivity and specificity as we will cover in this paper. Eventually we tested our system by playing back those extracted notes by piano and then recognize that music with human ear. Although there is still a lot to do for this subject, the primary results were quite impressive and promising.

2. Dataset and Preprocessing

First we needed some polyphonic piano songs in a sound file (wav) along with the corresponding list of notes so that we could use supervised learning algorithms for classification. One good option is to use midi files which contain the information about all the notes played in a song: their pitch, the exact timing when they are played, their duration and even their velocity, although we did not use this last item. Using a 'soundfont' associated to an instrument (in the rest of the text we used the same piano 'soundfont'), it is then easy to produce the wav file corresponding to these notes and to train the algorithm based on that. This rendered wav file is used as our observations, and the information contained in the midi file as our ground truth.

One famous dataset of polyphonic piano songs is MAPS (2) so we also decided to use it in this project. From the MAPS package we chose 60 songs in the `MAPS_ENSTDkAm_2` and `MAPS_SptkBGAm_2` folders.

We also used Ken Schutte's Matlab package to work with midi files (3). This package takes a midi file and parses it so that we can have access to the notes in a more friendly way. We produce the feature vectors by slicing the song wave into 100 ms intervals and for each interval we take the FFT. This makes sense because the pitch of a note is highly correlated with its frequency, so we would expect the FFT to give us more information on the pitch than the actual signal. For our learning algorithm we actually do not care how loud or low is the note played, so we can normalize the energy spectrum (4). For that matter we take the FFT of each 100 ms section and then calculate the norm of the FFT vector. For normalizing we need to divide this value by the sum of the vector elements. But there are many small elements in the energy spectrum that actually do not matter and only clutter the data. so we apply a threshold equal to 10% of the maximum value of energy spectrum and then we normalize it to one. The figure 1 shows the raw feature vector and the processed one. The sampling rate is equal to 44.1 KHz.

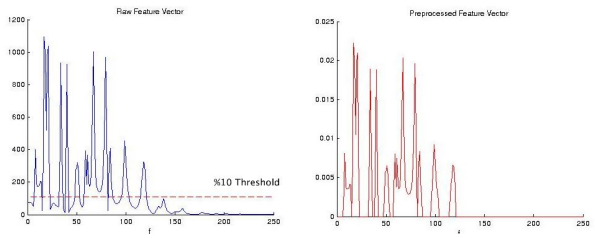

Figure 1. Thresholding applied to the feature vectors

3. Problems of dimensionality : PCA

Now that we have a full dataset, we can start processing the data. The basic idea to solve our problem is to view it as a classification problem. Each segment of 100 ms has a certain number of notes that are playing during that interval. Thus, we can see the output as binary with respect to one note : either this note is played, or this note is not played. If we train a binary classifier for each note, we should be able to tell for each segment if this note is played or not. Putting the information for each of the classifiers together, we could tell the list of notes that are played during that interval.

The first problem that we have here is that we have a really huge amount of data. We use half of the total data as our training dataset and it already includes more than 80000 intervals of 100 ms. Moreover, our feature space is also very large. Given that we had 4410 coefficients after applying the FFT to each segment, our space is $\mathbb{R}^{4410}$. It means that we have to deal with very large matrices and that most of the classification methods that we know will not work as such. This is why the first step that we will apply to our data is a PCA, so that we can drastically decrease the dimensionality of our feature space.

We run that PCA on a randomly sampled subset of 20000 examples. This number was chosen because it makes the PCA run quite quickly, in just a couple of minutes, while still being representative of the whole subspace. The justification behind the PCA is that since there are only a limited number of notes, we could expect the data to lie on a much smaller dimensional space than the initial one.

Figure 2 shows the log-magnitude of the singular values of the data matrix used for the PCA, sorted decreasingly. As we can see, there is a jump after 700 singular values, which verifies our assumption that the feature vectors lie on a smaller subspace.

We restrict our space to no more than a few hundred dimensions, which allows us to run logistic regression. For the logistic regression, we can then apply Newton's method, which reaches convergence after very few iterations. The reason behind that is that because of the PCA, the Hessian matrix is still small enough to be inverted fast, and running a gradient descent was much slower.

In later versions of our algorithm, we noticed that all 700 dimensions were not needed and we settled for a choice of 300 features kept after the PCA. This is an empirically supported choice as we can see in figure 3, where we plotted the sensitivity of a run our full

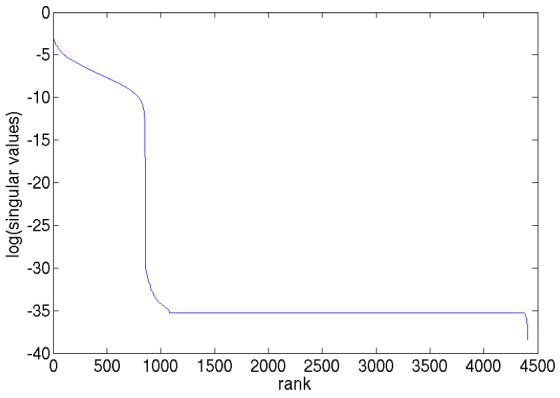

Figure 2. Singular values of the data matrix

algorithm (the next steps are described later) for the 45 notes that are played the most. As the number of features grows, the sensitivity increases too, which makes sense, but we can see that we get diminishing results and after around 100-200 features, there is not much improvement anymore. We get exactly the same kind of plots when we look at the specificity. We fixed our number of features used to 300 (the dimensions associated to the 300 largest singular values), which still gave a good increase in speed while training our algorithm without decreasing the performance.

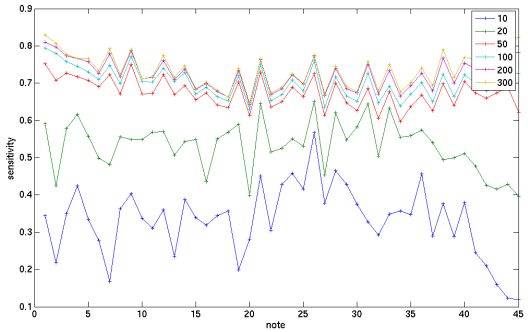

Figure 3. Sensitivity of the 45 most played notes with different dimensions of feature space

4. Dealing with unbalanced sets

If we look at figure 4, which shows the appearance frequency of the notes in the data that we set aside for training, we can see that all the notes are not equiprobable, and the notes in the medium range appear more often than the lower and higher notes. Some of the lowest and highest of the MIDI range did not even appear in our data, or very rarely. This fact may seem obvious but it means that we cannot deal with the

notes the same way if we want accurate predictions. Even the most frequent note only occurs in 15% of the samples, so if we do not try to correct the balance, our classification may be biased towards negative examples, and this may considerably decrease our sensitivity, since many positive examples will be misclassified.

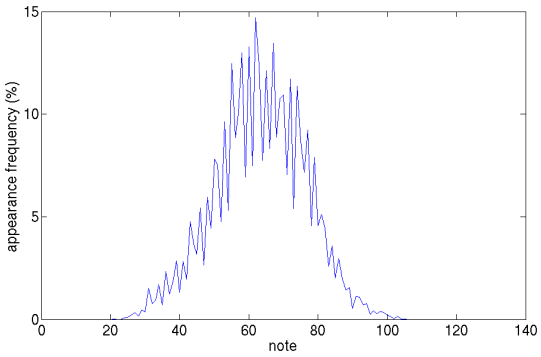

Figure 4. Appearance frequency of the notes in the half of the dataset used for training

The problem of unbalanced datasets has many applications and has been studied many times in the litterature. There are two basic ways to deal with this problem : sub-sampling and over-sampling.(5) If we have many more negative examples than positive examples, sub-sampling implies that we construct our training set by taking all the positive examples, but by sampling only a fraction of negative examples. In over-sampling, we take all our negative examples and we add several instances of the positive examples to balance the training set. These two methods have been shown to be asymptotically equivalent. In our case, sub-sampling (illustrated figure 5) looks more appealing because we already have a large amount of data, so it is not a problem to reduce it by sub-sampling the negative examples.

Using this technique, we can construct a **different training set for each note**, in which that note will be present in a fixed ratio of training examples. This is very useful for getting comparable performances for different notes, which was not the case before : our performances dropped as the note became less frequent in the training data. By adjusting the ratio of positive examples, we will see in our next section that our algorithm can perform differently. Also, we chose to address only the notes that appeared more than 2000 times in the totality of our training data, because we do not have enough information about the other notes. This only rules out 8% of the notes that appear in our dataset (in terms of number of intervals), which we decided was negligible in our application. It is important

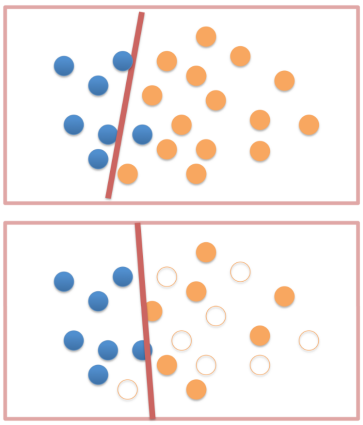

Figure 5. The boundary will be shifted depending if we use all the negative examples or only a fraction of them (sub-sampling)

to note that this limitation could easily be avoided if we had more data for these notes that appear less frequently.

5. Two different approaches in sub-sampling

5.1. Standard method

The first approach that we decided to take was to use sub-sampling at a 'low effect' setting : we just used it to equalize the ratio of positive examples in each of the training sets corresponding to the notes. More explicitly, since the most frequent note appears in 15% of the intervals, we can use this ratio for the other notes so that they also appear in 15% of the intervals of their training sets. The expected advantages of this method is that we still keep many negative examples so we still expect to have a high specificity. The inconvenient is that 15% is still quite low, and our classifiers may be biased towards negative examples, which will decrease the sensitivity.

In practice, we observe exactly these effects on our testing set, as we can see qualitatively on the piano-roll corresponding to this method applied to a small part of our testing set (figure 6b). To assess the performance quantitatively, we use specificity and sensitivity because they are good quantities for understanding how well we classify negative examples (specificity) or positive examples (sensitivity). This will be used consistently for the rest of the report. For this standard method, we get a specificity of 97.50% and a sensitivity of 71.45%.

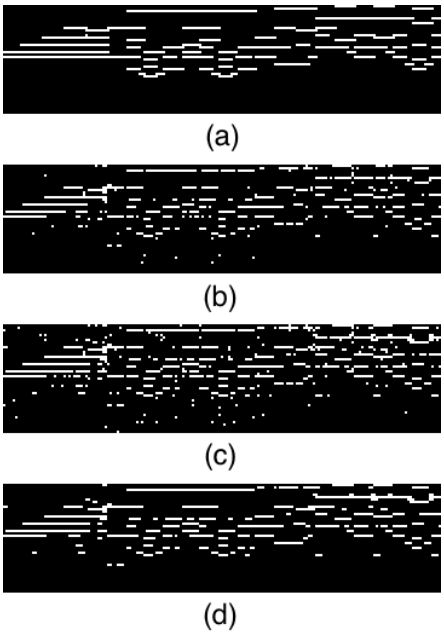

Figure 6. Data put in piano-roll shape : each line corresponds to the timeline of one note and each pixel on the horizontal axis corresponds to a different interval of 100 ms. A white pixel symbolizes the presence of that note in the interval while a black pixel symbolizes its absence. These piano-rolls correspond to the same part of a sound file and are respectively, from top to bottom : (a) Reference data (ground truth) ; (b) output of the standard method ; (c) output of the conservative method without post-processing ; (d) output of the conservative method with post-processing.

5.2. Conservative method

A second approach is to use sub-sampling at a 'higher effect' setting : we push the ratio of positive examples higher, at 20% instead of 15%. This means that we will tend to label more intervals as positive, so we will get a higher number of false positives (lower specificity), but also a lower number of false negatives, which means the sensitivity will be improved. If we look at the piano-roll of this method (figure 6c), we can see that this conservative approach makes the output very cluttered, which confirms our intuition.

Up to now, we have only treated the feature vectors corresponding to each interval as independent, and we can expect that adding a constraint on consecutive intervals may give us better results. This is what we attempt with the output of this method, as a post-processing step. We call $x_0 \in \mathbb{R}^n$ the binary vector corresponding to the intermediate (cluttered) result for one note (one line of the piano-roll). We want to find the binary vector x that is close enough to x_0 but that minimizes the number of transitions. This corresponds to the multi-objective problem of minimizing

$$J = \|x - x_0\|_2^2 + \mu \|Dx\|_2^2$$

where D is the square matrix with -1 on its diagonal and 1 on its upper second diagonal so that Dx returns the difference of consecutive elements of x, and μ is a parameter that chooses the relative weight between our two objectives.

The norm should be the l1-norm but it is equivalent to the squared l2-norm since our vectors only have values in $\{-1, 0, 1\}$. This problem is not easy to solve if we constrain x to a binary vector but we can solve this problem easily by relaxation : we solve in $\mathbb{R}^n$, and then we threshold to get a binary result. We can even solve it very fast if we consider x as a circular vector because in that case we can express our problem as $h_\mu * x = x_0$ where h_μ is a n-dimensional vector depending on μ and $*$ is the circular convolution, and then finding x is easy by taking the FFT of the expression above.

Figure 6d shows the result of this post-processing step using 6c as the input, for $\mu = 1.78$. We can see that we can get rid of many of the isolated false positives, while still keeping the true positives. On figure 7 we show the mean specificity and sensitivity associated to different values of μ. For very low values of μ, the post-processing step has no action. For very high values of μ, transitions are strongly penalized and the best move is to take a zero x. In between, there seems to be an optimum for $\log_{10} \mu \in [-0.1, 0.3]$ where the sensitivity slightly decreases but the specificity decreases considerably. This is the zone where we remove the outliers but not too many of the true positives, and so we take a value in this interval for our postprocessing step (after confirmation by testing and verifying we chose $\log_{10} \mu = 0.25$).

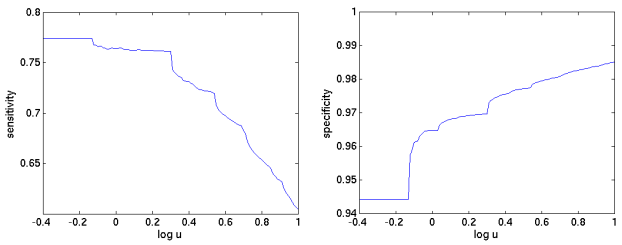

Figure 7. Sensitivity (left) and specificity (right) for different values of μ (logarithmic scale).

For this method, we get a specificity of 96.92% (comparable to the standard method) but an improved sensitivity of 76.12%. In the end, this method seems to

be more promising if we want to emphasize on having a higher sensitivity whithout sacrificing specificity.

6. Generalized method and final results

In fact, by choosing a different value for the ratio of true positives when we sub-sample, we can get different performances. If we prefer to sacrifice sensitivity in order to have fewer false positives, another option is to use a lower ratio, like the one we used in the standard method and to apply the post-processing. This gives a specificity as high as 98.81%, but a sensitivity of 68.59% only. In fact, by tuning this ratio as a parameter, we can span a whole trade-off curve given in figure 8.

We can see here that the post-processing gives in fact better results in both sensitivity and specificity, if we adjust the ratio of sub-sampling accordingly. The points corresponding to the two methods discussed in the previous part are circled.

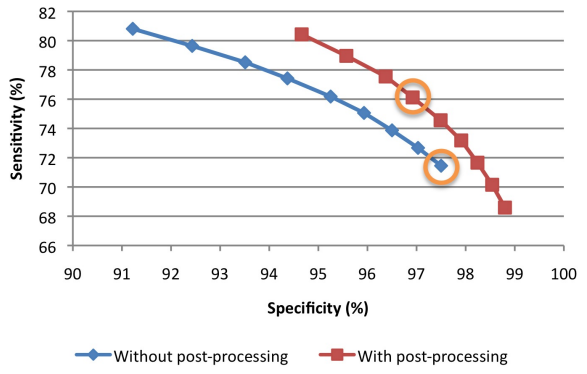

Figure 8. Trade-off curve of our two performance measures by taking different ratios in the sub-sampling step, from 15% (point on the right of each curve) to 23% (point on the left)

Qualitatively, when we listened to our actual results, we found that it is better for the ear to be on the higher-specificity / lower-sensitivity side of the trade-off curve : getting rid of false positives is much more important than recovering all the notes in their full length because our brain can easily reconstruct the missing parts, while the outliers can be heard very easily.

7. Conclusion

It was interesting to develop a full processing pipeline for this algorithm, because we could deal with many different aspects of machine learning, from data selection to classification or error measurement. Because of our time limitations, we could not try as many of our ideas as we would have wanted for each section of the algorithm and we focused on having a full algorithm working. Different ideas that we could have added to make our algorithm even better included : trying a different classification algorithm, like a SVM ; improving our time analysis based on the fact that notes are usually played on a certain tempo ; using a prior probability on the appearance of the notes using the key used in the song ; etc. However, on the whole, the output of our algorithm sounded very similar to the original music, and we were quite happy with the results that we got.

References

[1] N. Boulanger-Lewandowski, Y. Bengio and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," *ICML*, 2012.

[2] V. Emiya, R. Badeau and B. David, "Multipitch Estimation of Piano Sounds Using a New Probabilistic Spectral Smoothness Principle," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 6, pp. 1643-1654, 2010.

[3] http://www.kenschutte.com/midi

[4] J. Nam, J. Ngiam and H. Lee, "A Classification-Based Polyphonic Piano Transcription Approach Using Learned Feature Representations," *ISMIR*, pp. 175-180, 2011.

[5] H. He and E. Garcia, "Learning from Imbalanced Data," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1263-1284, 2009.