BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE ROMANIAN

DIPLOMA THESIS

Musical Pitch Detection Using Machine Learning Algorithms

Scientific Supervisor
Asist. univ. dr. Bocicor Maria Iuliana
Prof. univ. dr. Czibula Gabriela

Author
Ghiuruțan Bianca

2014

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ ROMÂNĂ

LUCRARE DE LICENȚĂ

Identificarea notelor muzicale folosind tehnici de învățare automată

Conducător științific
Asist. univ. dr. Bocicor Maria Iuliana
Prof. univ. dr. Czibula Gabriela

Absolvent
Ghiuruțan Bianca

2014

Table of Contents

Table of Figures

List of Tables

Introduction

The classification of sounds into notes has been one of the key moments in the history of music. Even though some musicians play by ear, playing by score is a recommended, widely-used technique. Music transcription, the process of transforming an audio recording of a song in a musical score (or sheet music) is, therefore, of tremendous help in assessing the quality of a musician's instrumental or vocal piece. The process of music transcription is conducted with great efficiency by many skilled musicians, but there are a few factors that make it quite an inconvenience: the long duration of the process, the fact that humans are more prone to error and the prevalence of audio illusions. These are the reasons why research regarding automatic music transcription is increasingly important.

The problem of identifying musical notes in a musical piece can be reduced to identifying pitches of sounds. In this paper we are focusing on applying some machine learning algorithms for detecting such pitches. Machine Learning [1], a branch of artificial intelligence, refers to the development of algorithms that enable computers to learn from experience. It uses concepts from artificial intelligence, mathematics, probability and statistics, information theory, neurobiology and other fields to develop solutions for several very important and complex problems, such as: medical diagnosis, speech and handwriting recognition, computer vision, natural language processing, etc. There are three main types of machine learning: supervised, unsupervised and reinforcement learning. Supervised machine learning is used for problems in which the output values are known for a set of input data, called the training examples. The main objective of supervised learning is to find a general function from these particular training examples that will be able to render an output for real input data. As opposed to supervised learning, in unsupervised learning, the training data is unlabelled. The plan of the algorithm is to organize and find patterns in the input data in such a way that it will be able to categorize future data as well. Reinforcement learning is the type of learning that refers to how algorithms should take certain actions should in order to maximize their reward. The actions taken are output values and the rewards are scalar evaluations. Reinforcement learning is specifically recommended in situations which include a long-term vs. short term reward.

In this paper we apply supervised learning techniques such as Support Vector Machine, Stochastic Gradient Descent Classifier and K-Nearest Neighbours algorithms for the classification of sounds into musical notes. In the context of Supervised Machine Learning, *the musical note identification problem*, or, equivalently, the pitch detection problem can be stated as follows: given a musical piece, fragmented in a certain number of time frames, classify each such time frame in one of several classes of pitches. As each pitch uniquely corresponds to one musical note, following the classification process, the musical note for each time frame of the piece will be known and consequently the musical score will be completely determined.

Each of the above mentioned classifiers was used and the experiments were made on a dataset composed of several musical pieces. The results of each algorithm will be presented and analysed. Furthermore, we offer comparisons between the results obtained with each technique.

The first chapter presents a few important notions about automatic music transcription. It starts with highlighting and explaining a few relevant music and physics terms and then displays a literature review, making a presentation of the efforts that have been made in this field in the recent years using machine learning algorithms.

The second chapter describes the machine learning algorithms used in the application: Support Vector Machine, the Stochastic Gradient Descent Classifier and K-Nearest Neighbours. It starts by presenting the general concepts behind these algorithms and then gives a detailed view on each one.

Chapter 3 presents the experiments, which have been performed with the help of the Scikit Learn library, a machine learning library for Python. It also describes the considered dataset, the features that were extracted for it and the way the data processing has been made. The metrics used for the evaluation of each algorithm are briefly described in this chapter as well. The paper then proceeds to present the results of each algorithm, using several evaluation metrics. Comparisons between the performances of each method, with respect to the metrics that are used and to the computation time are pointed out, as well as discussions about the obtained results.

In the end, this paper outlines the main conclusions of this study, determining that machine learning approaches to music transcription and pitch detection represent a fast, cheap and efficient alternative to the classical methods.

Chapter 1: Automatic music transcription

1.1 Music - Basic Theoretical Concepts

Notes and *octaves* are a way of organizing and classifying sounds. A note is the *pitch* (or frequency) of a sound, but can also be considered a symbolic notation of such a pitch. A *musical score* is the chaining of the notes of a song, in the order they appear, with respect to pauses. In Figure 1.1 an example of such a musical score is shown.

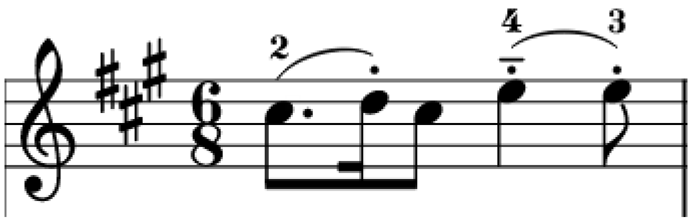
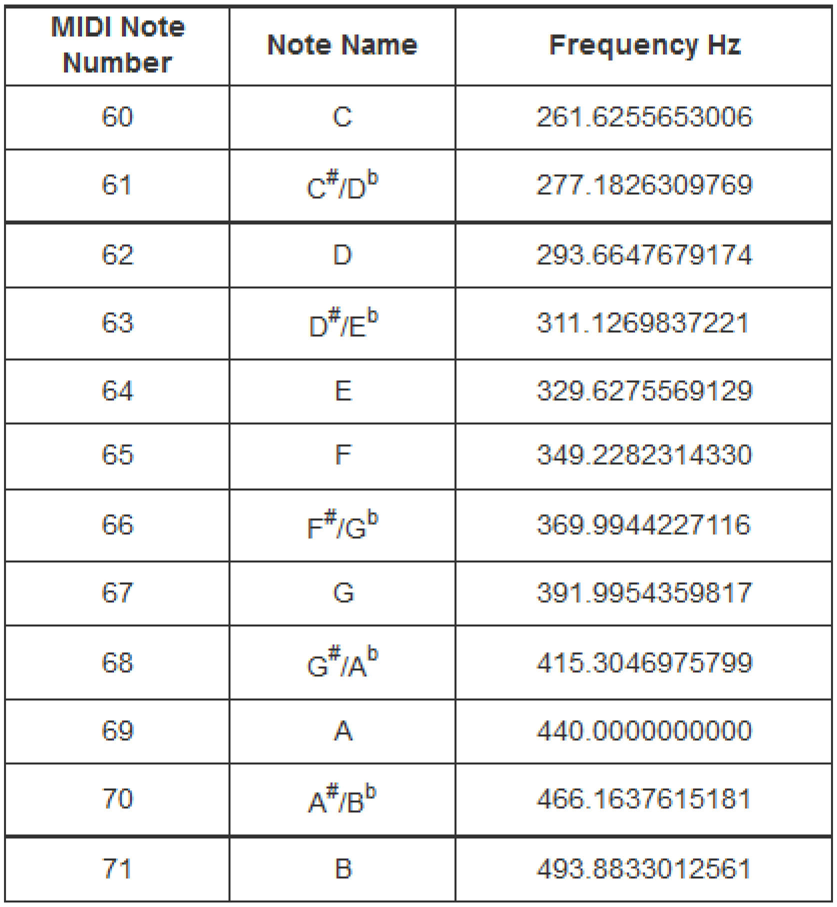


Figure 1.1: Example of musical score. Figure source [2].

An octave is the interval between two musical pitches, the first one having the value of the frequency half of the other one. There are 11 octaves and each one of them contains 12 semitones (the interval between neighbouring notes), except for the last one, which has only 8. In total, there are 128 different notes over 11 octaves.

A pitch class is the collection of notes that are an octave apart. For example, the pitch class C contains all the C notes in all octaves. The quality of a pitch is referred to as a *chroma* and all pitches in a class share the same chroma.

Another important notion is the pitch represented in MIDI number. MIDI (acronym for Musical Instrument Digital Interface) is a standard for electronic music devices (e.g. synthesizer) meant to provide certain information useful in rendering the sound of that particular piece of music. It also holds data such as: the pitch value, the length or the volume of a note. In figure 1.2 an example of a MIDI pitch value for the first octave (with 440Hz tuning) is shown.



MIDI Note Number	Note Name	Frequency Hz
60	C	261.6255653006
61	$C^{\#}/D^{b}$	277.1826309769
62	D	293.6647679174
63	$D^{\#}/E^{b}$	311.1269837221
64	E	329.6275569129
65	F	349.2282314330
66	$F^{\#}/G^{b}$	369.9944227116
67	G	391.9954359817
68	$G^{\#}/A^{b}$	415.3046975799
69	A	440.0000000000
70	$A^{\#}/B^{b}$	466.1637615181
71	B	493.8833012561

Figure 1.2: An example of the MIDI pitch values and the frequency of notes. Figure source: [20]

In order to obtain the transcription of the song, each note of the song must be identified. A note has a corresponding *fundamental frequency* (F0). This is also the frequency that corresponds to the pitch that is identified by the human ear.

An important categorization of signals is between the monophonic sounds and the polyphonic sounds. Monophonic sounds come from a single instrument (or voice) at a time. Polyphonic sounds refer to an instrument playing multiple notes at once or multiple instruments playing at once, thus rendering simultaneous notes. This paper will only discuss pitch detection for monophonic sounds.

A note also has overtones and harmonics. An overtone is any frequency that is higher than the fundamental frequency of a sound, while harmonics are an integer multiple of the fundamental frequency.

In order to obtain the best possible results, multiple features of the audio signal must be taken into consideration. These are split into two categories: *time-domain features* (zero crossing rate, autocorrelation, YIN metrics etc.) and *frequency-domain features* (FFT, STFT, spectrogram, cestrum metrics, spectral flux etc.). The latter ones are the most used in machine learning algorithms for pitch identification. An FFT (Fast-Fourier Transform) is a feature that computes the DFT (Discrete Fourier Transform) and its inverse. This is achieved with the use of the Fourier analysis, which is a method of converting from the time domain to the frequency domain.

1.2 Literature review

There are several papers in the literature that present research that has been done on this subject. Most pitch detection algorithms (and all presented here) are use classifiers from the field of machine learning. In the following three approaches from the literature will be presented.

Deshpande [3] approached the musical pitch identification problem using various machine learning based techniques implemented by the Java library Weka [4]: Naïve Bayes, Support Vector Machine (SVM) and Logistic Regression.

Due to the performance challenge given by the 128-ways classification, the author decided to classify only the pitch class, thus having only 12 classes. The dataset he used included popular guitar songs, tracks that were then sampled at 8 KHz. The size of the training examples was about 40 000. Since the aim of his application was to ease the process of assessing a student's performance, it was important that the training data contained a representative amount of accidentals. After experimentation and literature research, the author chose the spectrogram as a feature. The first algorithm the author experimented with was the Naïve Bayes. The best accuracy this technique reached was approximately 64%. One of the reasons the accuracy isn't higher is because the Naïve Bayes method assumes that the features are independent of each other, which in this case is definitely not true. The second used algorithm is the Logistic Regression, which showed a good accuracy of approximately 74%. The RoC, Precision and Recall [5] performances were also better than those of Naïve Bayes. The third one was the SVM (with Sequential Minimal Optimization), which not only provided better accuracy but it did so on a smaller training set. Increasing the training set size did not show significant improvement.

Nevertheless, the Logistic Regression algorithm showed better RoC, Precision and Recall characteristics on a notably higher number of training samples.

In figure 1.3 the RoC Curve and the Precision-Recall Curve of all the algorithms for the notes C (chroma 0) and C# (chroma 1) are shown.

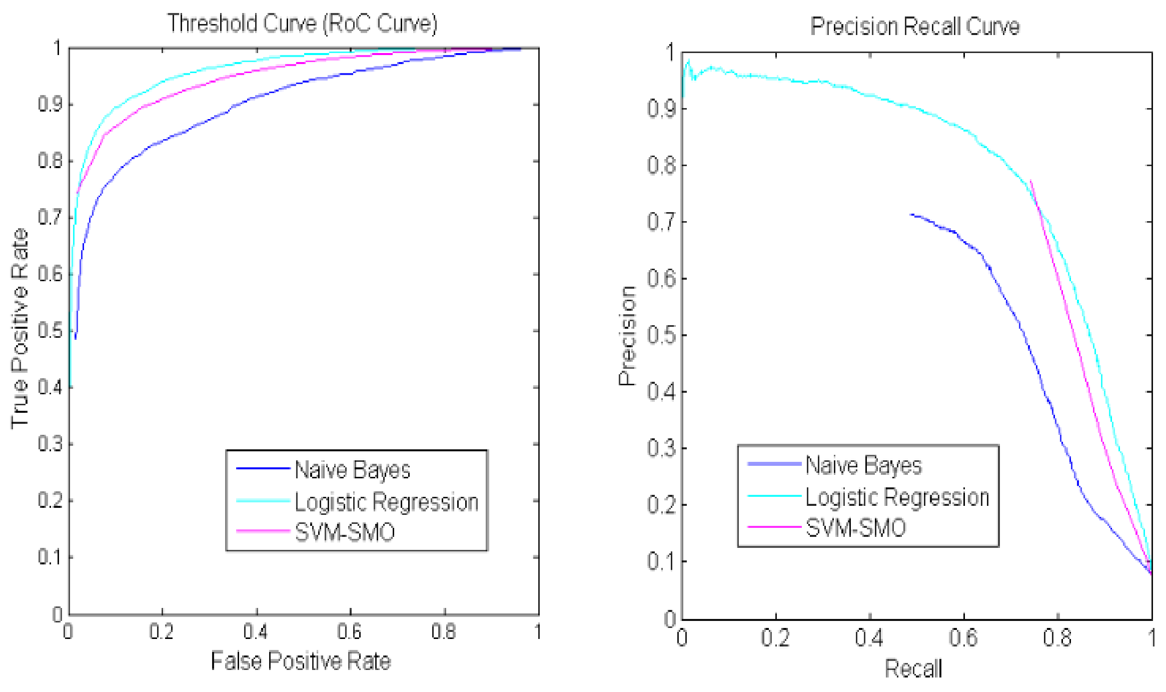
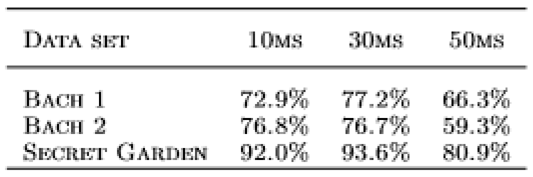


Figure 1.3: Naïve Bayes, Logistic Regression and SVM performance for the first two chromas. Figure source: [3] .

Kidson and Zhi [6] developed an application which reproduces violin scores through pitch identification. The feature they used was the spectrogram and the algorithms chosen were the Logistic Regression and Support Vector Machine. The authors chose 38 classes (each representing a note) that were overwhelmingly common in violin playing.

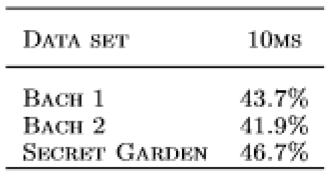
The Logistic Regression Classifier provided very good results. Choosing the window size for calculating the FFT proved to be a quite a challenge.



Data set	10ms	30ms	50ms
Bach 1	72.9%	77.2%	66.3%
Bach 2	76.8%	76.7%	59.3%
Secret Garden	92.0%	93.6%	80.9%

Figure 1.4: Performance of the Logistic Regression with multiple window sizes. Figure source: [6].

The SVM, though expected to obtain more impressive results, presented a significantly worse accuracy (figure 1.5). The implementation was based on the LIBSVM library. The data was scaled and the RBF kernel was used. Cross-validation was performed in order to find the most suited value for the used parameters (C and γ). The best number of features was chosen to be 184 (which is associated to a window size of 10ms). In these experiments as well, it was found that increasing the number of training samples did not help performance very much.



Data set	10ms
Bach 1	43.7%
Bach 2	41.9%
Secret Garden	46.7%

Figure 1.5: Performance of the SVM. Figure source: [6].

Graham E. Poliner also performed some experiments for the detection of the musical pitch [7]. The training samples were chosen in two ways: incremental sampling (taking a certain number of frames from the beginning of each excerpt) and random sampling. For musicology reasons (the frequency of a note must be the same for its entire duration), the first approach is capable of yielding better results than the second one for a small training set size. For big training set sizes, the accuracies of the methods converge. The author tried several more experiments with the dataset consisting of randomly selected frames from 13 resampled transpositions of the excerpts (a piece of music). The author experimented with an all-versus-all SVM which showed an accuracy of 67.7% for raw pitch identification and 72.7% for chroma identification (chroma doesn’t take into consideration the octave of the note). A one-versus-all SVM with a linear kernel showed an accuracy of 69.5% for raw pitch identification and 74.0% for chroma identification. The one-versus-all SVM with a RBF kernel obtained the best results: 70.7% for raw pitch identification and 74.9% for chroma identification.

1.3. Conclusions

This chapter presented a brief introduction into music theory and defined some basic concepts that need to be grasped in order to be able to study and understand automatic music transcription. Each musical note has an associated pitch, represented as a number and, in total, there are 128 such pitches. Therefore, each time frame of a musical piece may be categorised in one of the pitch classes, thus facilitating the application of supervised learning techniques for the classification of frames into classes of musical notes.

The second part of the chapter dealt with three approaches from the literature that addressed the problem of automatic music transcription. These approaches were chosen because they used supervised algorithms, some of which are the same with the ones that we will further use in this study. The approaches were described and the results obtained for various datasets were shown.

Chapter 2: Machine learning algorithms

2.1 Support vector machine

2.1.1 Introduction

Support vector machine (or *support vector network*) is a supervised machine learning algorithm mainly used for classification, but which can also perform regression. The original version of the SVM was invented in 1979 by Vapnik, but the current version (which uses soft margin) was only introduced in 1993, by Cortes and Vapnik and published in 1995 [8]. The support vector machine is a very well-known algorithm with great potential for solving real world problems; it has been used for applications such as handwriting recognition, text and hypertext categorization, classification of images, classification of proteins etc.

The SVM works by training on a set of training examples which have each been labelled as belonging to one of two classes, and is then capable of assigning a new example into one of these classes. Thus, support vector machine is a non-probabilistic binary linear classifier. The SVM represents examples as points in space and tries to map them so that there will be a clear gap between the examples from one class and the examples from the other. The wider the gap is, the lower the error of the algorithm. This so-called gap is formally known as a hyperplane (it can also consist of a set of hyperplanes). The optimal hyperplane is the one that separates the two classes perfectly and maximizes the distance between it and the closest training data point of any class (this distance is called functional margin).

Besides the linear classification, the SVM is also capable of performing non-linear classification, with the use of a kernel function. This kernel trick has the ability to map the training examples into a high-dimensional (even infinite) space. Figure 2.1 shows a representation in a two-dimensional space of a SVM.

Another advantage of the SVM is that its VC-dimension can be calculated, unlike for other machine learning algorithms, such as the neural network. A VC-dimension is an alternative to cross-validation that can estimate how well an algorithm will perform on unseen data, based only on the training error.

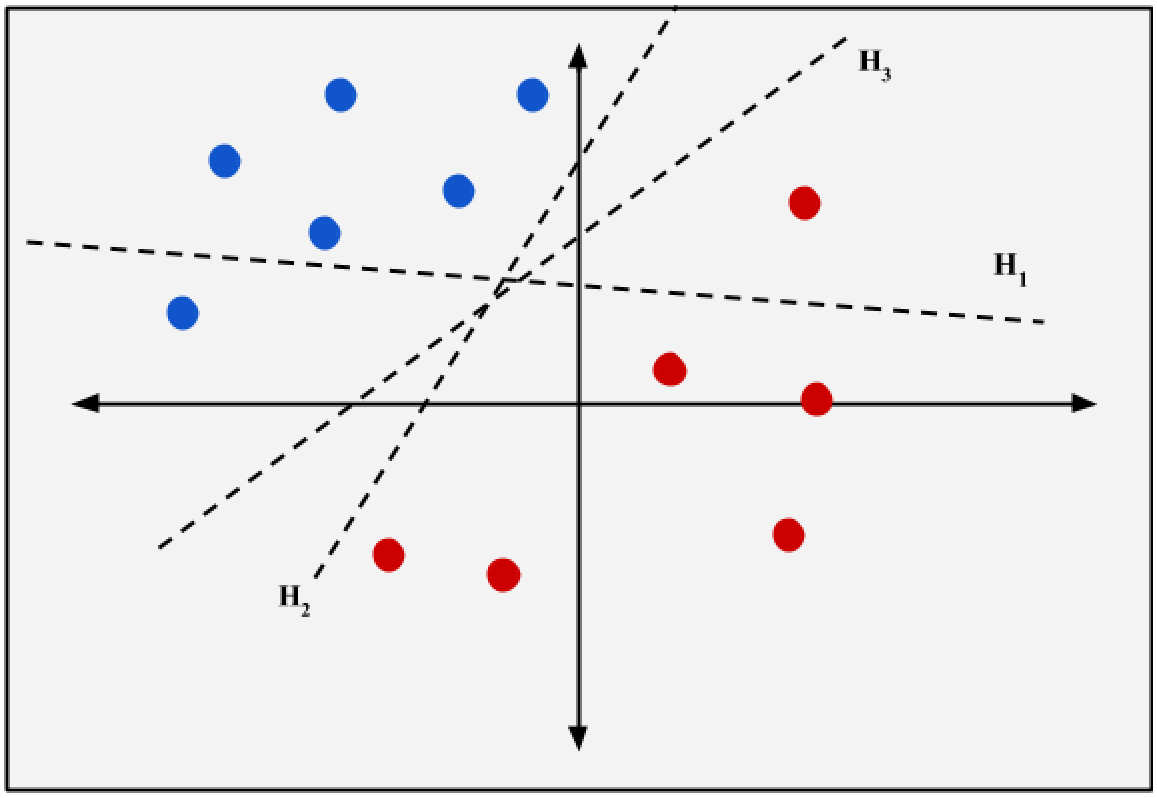


Figure 02.1 A two-dimensional SVM. H1 does not separate the classes well; H2 separates the classes, but has a small margin; H3 is the maximum-margin hyperplane.

2.1.2 Linear SVM

In the training dataset D we have n training examples. Let D be:

$$D = \{(x_i, y_i) \mid x_i \in \boldsymbol{R}^p, y_i \in \{-1, 1\}\}_{(i = 1)}^{n}$$

which means that each example has p inputs ($\mathbf{x_i} \in \boldsymbol{R}^p$) and the possible classes are -1 or 1 (since $y_i \in \{-1, 1\}$) where $\mathbf{y_i}$ is either 1 or -1 (representing the class of the point $\mathbf{x_i}$). Each x_i is a *p*-dimensional vector.

The SVM tries to find the maximum-margin hyperplane that separates the points having y_i =-1 from the ones having y_i =1. A hyperplane can be defined as a set of points $\mathbf{x}$ with the following property:

$$w \cdot x + b = 0$$

where $\mathbf{w}$ is the normal vector to the hyperplane, "$\cdot$ " is the dot product and $\mathbf{b}$ indicates the offset of the hyperplane from the origin. In case the inputs are linearly separable, two hyperplanes can be chosen to separate the data. The space between them is called margin, and any samples on the margin are called support vectors. If the distance between the two hyperplanes is maximized, the maximum-margin hyperplane can be determined.

The two hyperplanes have the following equations:

$w \cdot x + b = -1$ and $w \cdot x + b = 1.$

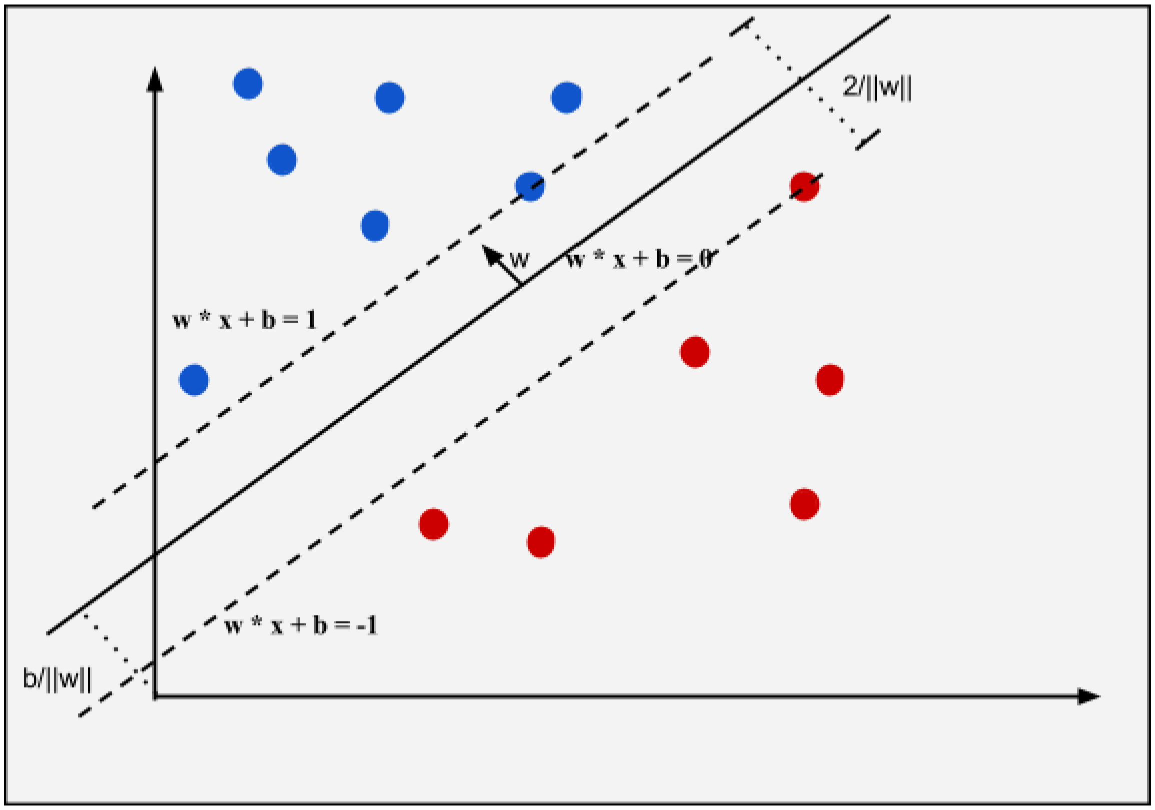


Figure 02.2 A depiction of a maximum-margin hyperplane and the two margins.

The distance between them is $\frac{2}{||w||}$, therefore the bigger the vector norm, the smaller the distance. Another condition is that the data points must be outside the margin, so the following constraint must be added:

$$y_i(w \cdot x_i + b) \geq 1 \text{ for all } 1 \leq i \leq n.$$

Since the norm of the vector involves a square root, a possible simplification of the problem can be done by substituting the norm of the vector with $\frac{1}{2||w||^2}$. The purpose of the ½ factor is just to facilitate the calculations. Thus, the following condition is obtained:

$$\arg min_{(w,b)} \frac{1}{2||w||^2}.$$

This constraint can also be expressed using the Lagrange multipliers α :

$$argmin_{(w,b)} max_{\alpha \geq 0} \left\{ \frac{1}{2} \|w^2\| - \sum_{i=1}^{n} \alpha_i [y_i(w \cdot x_i + b) - 1] \right\}$$

Since α_i must be set to 0, the points $y_i(w \cdot x_i + b) - 1$ are not important. Next, the problem can be solved by standard quadratic programming methods. According to the Karush–Kuhn–Tucker condition, the weights vector can be expressed as a linear combination of the training vectors.

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i.$$

When α_i will be greater than 0, the corresponding $\mathbf{x_i}$ will be a support vector (a vector situated on the margin which satisfies the equation $y_i(w \cdot x_i + b) = 1$). Therefore, $w \cdot x_i + b = \frac{1}{y_i} = y_i,$ since $y_i \in \{-1, 1\}$. This is equivalent to $b = y_i - w \cdot x_i$. To make the SVM as robust as possible, it is better to average over all support vector. Thus, we obtain: $b = \frac{1}{N_{SV}} \sum_{i=1}^{N_{SV}} [\![(y]\!]_i - w \cdot x_i)$, where N_{sv} is the number of support vectors. Thus, we have found both the weights (w) and the offset (b) and the maximum-margin hyperplane can be identified.

2.1.3. Nonlinear SVMs

In the case of non-linearly separable datasets, a nonlinear SVM can be created by mapping the training examples into a much higher-dimensional space. The goal is to facilitate the process of finding a maximum-margin hyperplane. A much better discrimination between datasets can be achieved with the use of a higher-dimensional maximum-margin hyperplane, even though the sets in the original space were not convex.

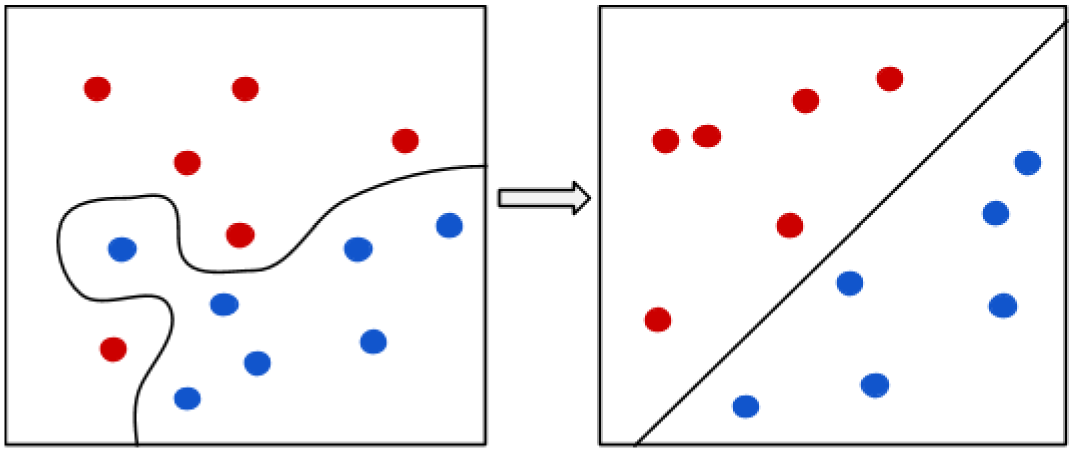


Figure 2.3 The dataset in the original space and then mapped in a higher-dimensional space. It can be observed that the dataset is linearly-separable in the higher-dimensional space.

The nonlinear SVM is created by replacing the dot product of the inputs and the weights with a kernel function. There are various kernel functions. Among the most used are the following [9]:

- Gaussian radial basis function (RBF): $k(x_i, x_j) = \exp\left(-\gamma \left\|x_i - x_j\right\|^2\right)$,

  γ must be greater than 0. γ is a tuneable parameter, but it usually has this form $\gamma = \frac{1}{2\sigma^2}$;
- Polynomial $k(x_i, x_j) = (x_i \cdot x_j + 1)^d$;
- Hyperbolic tangent $k(x_i, x_j) = \tanh(\kappa x_i \cdot x_j + c)$ for $\kappa > 0$ and $c < 0$.

The kernel is applied as follows: $k(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j)$, where $\varphi(x_i)$ represents the mapping of the feature vector into the higher-dimensional space. The weights w can then be expressed as:

$$w = \sum_{i=1}^{n} \alpha_i y_i \varphi(x_i).$$

Dot products between the weights and the transform $\varphi(x_i)$ are represented as follows:

$$w \cdot \varphi(x) = \sum_{i=1}^{n} \alpha_i y_i k(x_i, x).$$

The kernel trick is a very used and successful method, since datasets often prove to be non-linearly separable. Choosing the right kernel and the right parameters for an application can be a challenging task. One should analyse the problem type they're working on and also do literature research to see what has given the best results.

2.1.4 Multiclass SVMs

Though SVM is a binary classifier, multiclass classification can also be achieved with the use of SVMs. Usually, the solution to this problem is to divide the multiclass classification problem into multiple binary classification sub-problems. The most common way of doing this is by creating binary classifiers which separate one class from all the rest (method known as *one-versus-all*) [10] or binary classifiers that separate each two classes (method known as *one-versus-one*). In the one-versus-all approach, the classifier that has given the best results (the highest output) is chosen as the winner. This classifier will show the class it found. In the one-versus-one approach, the class that has been chosen the most (by all the classifiers) is the winner.

2.2 K-Nearest Neighbours

K-Nearest Neighbours (k-NN) [11] is an instance-based machine learning method. It is a non-parametric algorithm which can be used for classification and regression. In the case of classification, the algorithm decides the result class for a specific sample based on a majority vote of its *k* neighbours. The parameter *k* is user-defined; if it is equal to 1, the algorithm is called nearest neighbour and the sample is assigned the class of its nearest neighbour. Figure 2.3 shows an example of a k-NN classification, with different values for the parameter *k*.

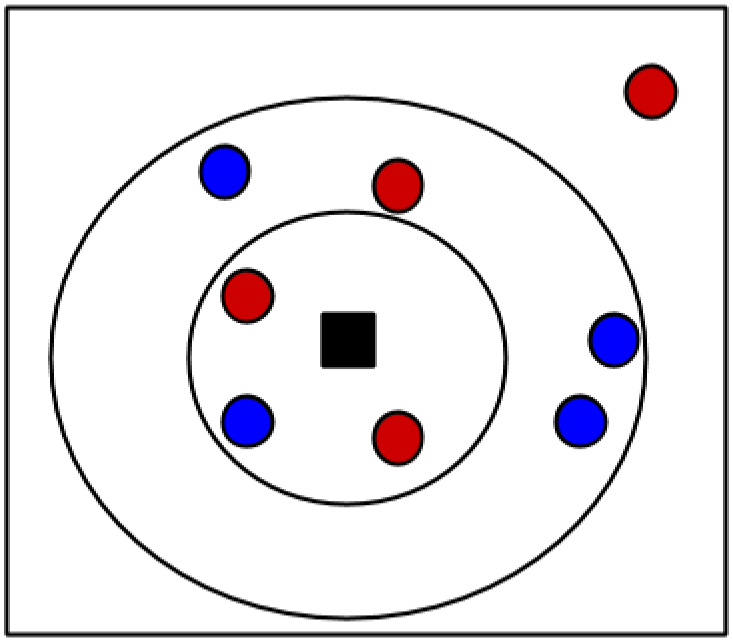


Figure 2.4 Example of k-NN classification. The black square must be assigned to the class of blue circles or the one of red circles. If k = 3, it will be assigned to the class of red circles, but if k = 7, the assigned class will be the one of the blue circles.

Some applications might work better if the contributions of the neighbours are weighted in such a way that the closest neighbours contribute more to the decision than the more distant ones. A frequent value of such a weight is *1/d*, where *d* is the distance from the test sample to the neighbour.

K-Nearest Neighbours is considered one of the most simple machine learning algorithms. A reason for this is the fact that it has no training phase, all computation being done at the moment of classification. However, the set of data (the neighbours) for which the class is known, is referred to as training set.

One of the disadvantages of k-NN can emerge when the numbers of samples from each class is highly different. In this scenario, the classes that have the highest frequency will often win the voting. One solution is to weight the contribution of the neighbours with the inverse of the distance (the *1/d* scheme). Another solution is the abstraction in data representation (with the help of a self-organizing map, for example). In binary classification or multiclass classification where the number of classes is even, it is a good approach to choose the *k* parameter an odd number, in order to avoid tied votes.

The value of the *k* parameter can be very important to the accuracy of the algorithm. A bigger value for k diminishes the effect of noise, but can also deteriorate the boundary between classes. There are several heuristics algorithms for choosing the best value for k, such as the hyperparameter optimization. Another important concern is the feature selection and feature scaling. K-NN can be greatly damaged by the presence of irrelevant features. One way to overcome this problem is by using evolutionary algorithms.

Often times the number of dimensions of the feature vector is very large. In these cases, dimensionality reduction is advised because it avoids curse of dimensionality. This can be done with principal component analysis (PCA), canonical correlation analysis (CCA) or linear discriminant analysis (LDA).

Data reduction is often advised in case of extremely large datasets. Usually, only a part of the data examples are relevant for the classification. They are known as *prototypes* and are detected in the following way: the class-outliers (the training data that is classified incorrectly) are selected and the rest of the data in split into two sets: one formed of the prototypes and the other one formed of absorbed points that can be correctly classified using those prototypes. The absorbed points can then be removed from the training to improve performance.

2.3 Stochastic gradient descent

Stochastic gradient descent (SGD) [12] is a gradient descent optimization algorithm that aims to minimize a loss function. In machine learning, the loss function is usually used to choose the best weights (parameters) for a function by computing the difference between the estimated output and the actual output (also known as error).

The SGD has given great results in problems such as natural language processing or text classification. Besides being very efficient, the SGD can easily be implemented. The drawbacks of SGD are the facts that it is sensitive to feature scaling and it requires a regularization parameter and a number of iterations.

The main difference between SGD and Gradient Descent (also known as Batch Gradient Descent) is the fact that SGD updates itself after each sample (with respect to that single sample), while Gradient Descent (GD) only updates itself after a complete iteration. Therefore, the SGD starts approaching the minimum with the first sample it goes over and then continues to improve itself with each new sample [13] [14]. This is the reason why the SGD is sometimes preferred to GD in the case of a very large dataset. The SGD algorithm is able to converge rapidly, but will not be able to minimize the error as well as GD. In the case of a fixed learning rate (usually denoted by α) the SGD will oscillate around the global minimum, but with a decreasing learning rate, the algorithm is more probable of converging to the global minimum. Even if the minimum is not reached, in practice, the approximations obtained by the SGD are usually accurate enough. The data is recommended to be shuffled on each iteration, to minimize the risk of overfitting.

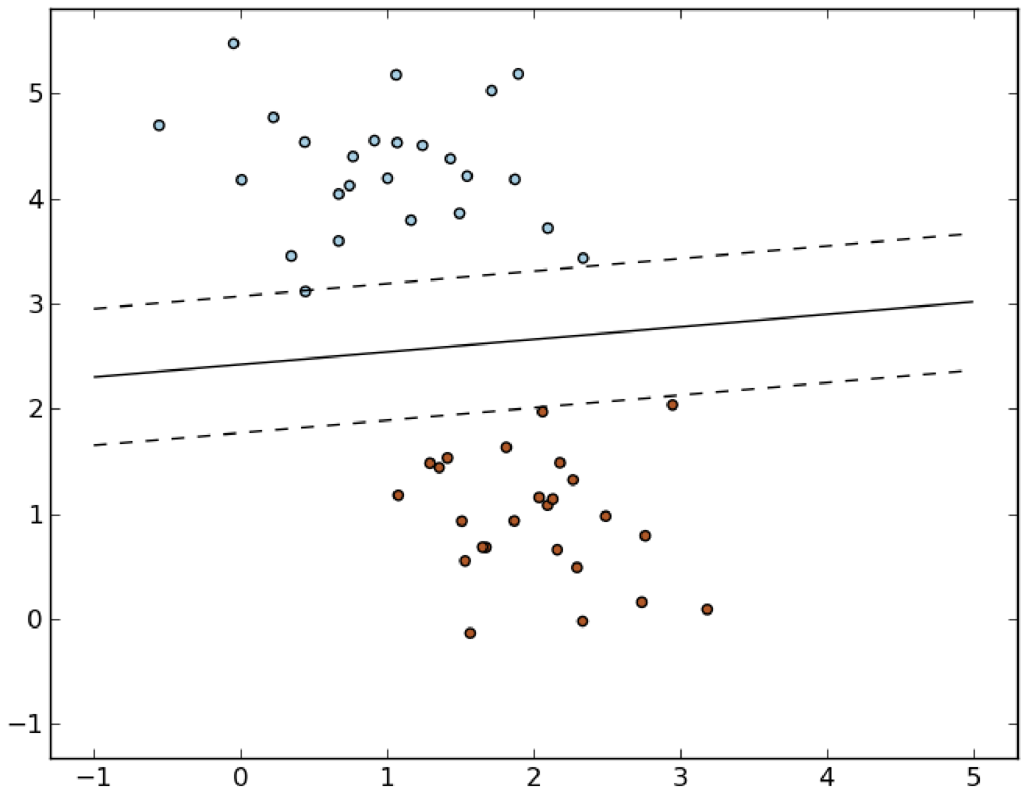


Figure 2.5 A maximum margin hyperplane separating a two-class dataset using linear SVM trained with SGD. Figure source: [14]

2.4 Conclusions

This chapter presented the machine learning algorithms that we used in our study of automatic music transcription. We focused on three supervised techniques, namely Support Vector Machines, K-Nearest Neighbours and Stochastic Gradient Descent.

The Support Vector Machines (SVMs) were presented along with their theoretical background. We described both linear and nonlinear SVMs and we briefly mentioned some of the kernel functions that are used for the nonlinear version. Given that the SVM is a binary classifier, we also mentioned about how multiclass classification can be achieved using multiclass SVMs.

We also presented an instance based learning technique, more precisely the k-Nearest Neighbour classification. We discussed about the different values of the *k* parameter and how these influence classification, as well as about various disadvantages of the algorithm and how such situations might be avoided.

The third method that was presented is Stochastic Gradient Descent (SGD), a technique used for function minimisation. The function is chosen in direct connection with the differences between the outputs and the target values. SGD was presented in comparison with Gradient Descent.

Chapter 3: Application – Detecting Musical Pitches using Machine Learning Algorithms

3.1 Application Development

This subchapter describes the application that was implemented to achieve classification of musical pitches, which uses the machine learning algorithms that were presented in the previous chapter.

The implementations of the used machine learning algorithms that were chosen for our experiments are offered by a learning library written in Python, namely the Scikit-learn library [14]. Therefore, we have also chosen the Python language for the development of our application.

The software application can be regarded as a *pipeline*, which starts with reading and processing the input chorale, given as audio recording in *wav* format and finishes with the classification of the pitches in the test chorale and with the display of the values for several metrics that we considered, which will be detailed in the next subchapter.

In the following, we will describe each step of the pipeline.

Firstly, the input chorales are processed to extract the features for the machine learning algorithms. After literature research, it has been found that the most relevant features belong to the frequency-domain and one of the most import features in this regard is the FFT (Fast-Fourier Transform). To extract this we used an efficient audio features extraction library called YAAFE [16]. This library provided the feature called magnitude spectrum, which renders the FFT in its absolute value for each frame, using the Hanning analysis window [17]. The frame size and the hop size (parameters of the function that extracts the features) had to be chosen with respect to the frame and hop size of the songs. Different number of frames sizes will give different number of features for each input, thus affecting the dimensionality of the data. If it is possible, the frame size should be chosen in such a way that it will hold enough data, but not too much to be vulnerable to the *curse of dimensionality*. The result of this first step consists of *csv* (comma separated value) files, in which each row represents a frame (an input instance) and the columns represent the features. The last column is the target value, more specifically the ground-truth pitch values (in MIDI number) for each frame. The targets are normally given as input, along with the chorales. Table 3.1 illustrates an example of file that is obtained as a result of this step.

3.41	1.75	0.89	0.69	0.47	0.44	…	0
4.73	0.68	0.69	0.46	0.21	0.31	…	0
4.69	0.32	0.22	0.30	0.12	0.04	…	0
5.80	0.43	0.17	0.11	0.14	0.04	…	48
5.08	0.20	0.09	0.20	1.11	1.17	…	48
5.08	0.08	0.08	0.09	1.43	2.24	…	…
4.76	0.23	0.10	0.52	2.43	2.75	…	55

Table 3.1: Example of file obtained after data processing.

The files obtained this way can afterwards be used for the training and testing of the classifiers. The second step consists of the Python application that was implemented to instantiate and use the selected three machine learning algorithms offered by Scikit-learn. The classes corresponding to each algorithm are as follows:

- for the SVM classifier the class is called SVC;
- for the k-NN algorithm the class is called KNeighborsClassifier;
- for the SGDClassifier Scikit-learn offers a class called SGDClassifier.

The interfaces of the data types implemented in these classes allow us to use them without much difficulty. All we need to do is to instantiate them and choose a set of values for the specific parameters of each one.

The application allows the user to choose the algorithm to be applied for testing. After the testing is done, the results of the evaluation are printed. Using instruments provided by the same library, we are able to determine the performances of the classifiers. We compute the values of several evaluation measures (which will be described in Section 3.2.2) and generate a classification report which are displayed to the user.

3.2 Experiments and Results

This subchapter presents the experiments that we designed for the detection of musical pitches. We begin by describing the dataset that was used and then continue to present the evaluation measures we employed for the comparison of the algorithms' performances. The obtained results for each algorithm are presented and analysed and in the end we also offer a comparison of the obtained results and we discuss the algorithms' differences in performance.

3.2.1. Dataset

The data was taken from a versatile music dataset [15], which consists of 10 chorales belonging to Johann Sebastian Bach. Ideally, all of these chorales would be used in our experiments, but unfortunately, the computer the experiments were conducted on does not have enough RAM to memorize so much data. Therefore, only four of them were chosen. These were: *Ach Gottund Herr*, *Ach Lieben Christen*, *Christederdu Bist Tag Und Lichtand* and *DieSonne*. The first two were used for training, the third one was used in the validation phase as test data, and the fourth one was used in the final test. In each of them the following instruments are played: violin, clarinet, saxophone and bassoon which correspond to Soprano, Alto, Tenor and Bass. For each such part, the audio recording is provided, which was rendered in *wav* format. The wav format is usually preferred in audio applications because it contains uncompressed audio, thus the quality being higher.

The dataset also provides the ground-truth pitch values (in MIDI number) for each part (violin, clarinet, saxophone and bassoon) in each time frame. These pitch values formed the output data. Some of the frames had the pitch value 0, which represents the lack of a note (e.g. a rest - pause in music). Figures 3.1 - 3.4 represent the musical notes (pitch values) for each time frame in the training set, for each of the four instruments. These charts are intended to show the range of notes for each instrument. It can be observed that the pitches for violin range between 65 and 80, for clarinet between 60 and 70, for saxophone they start lower, at 50 and go up to 70 at most and even lower for bassoon, from 40 to 65. This information could be used in the classification process, if the classification were to be made separately for each instrument.

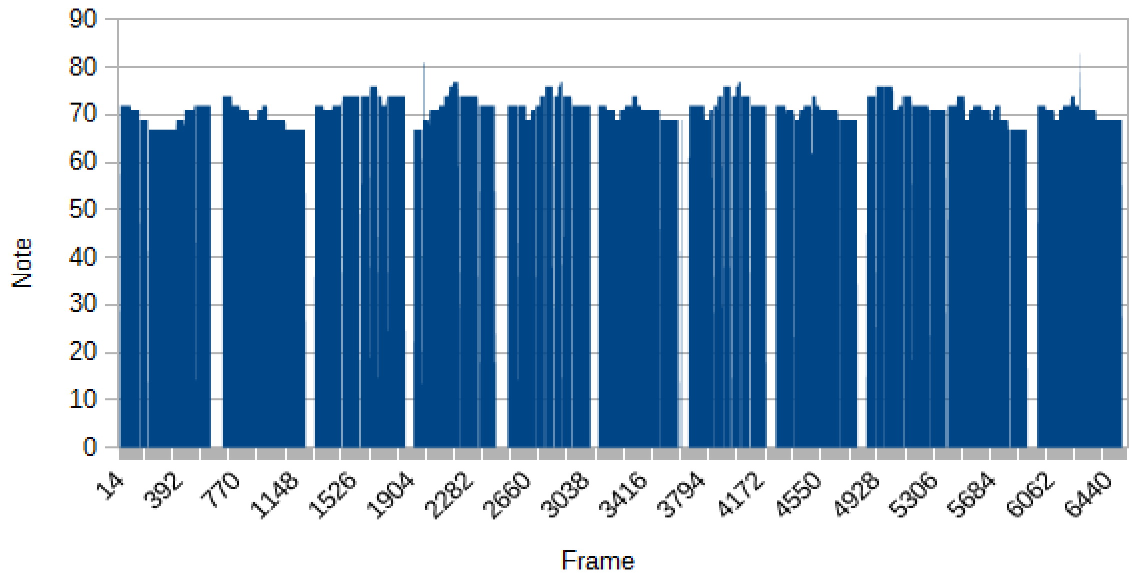


Figure 3.1 The notes for violin in the training set.

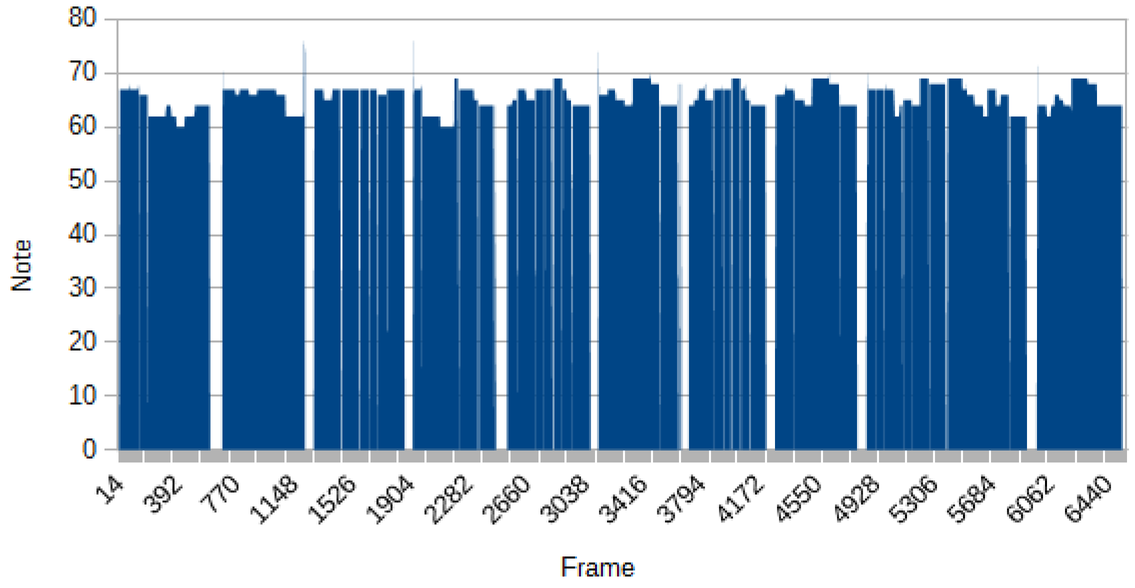


Figure 3.2 The notes for clarinet in the training set

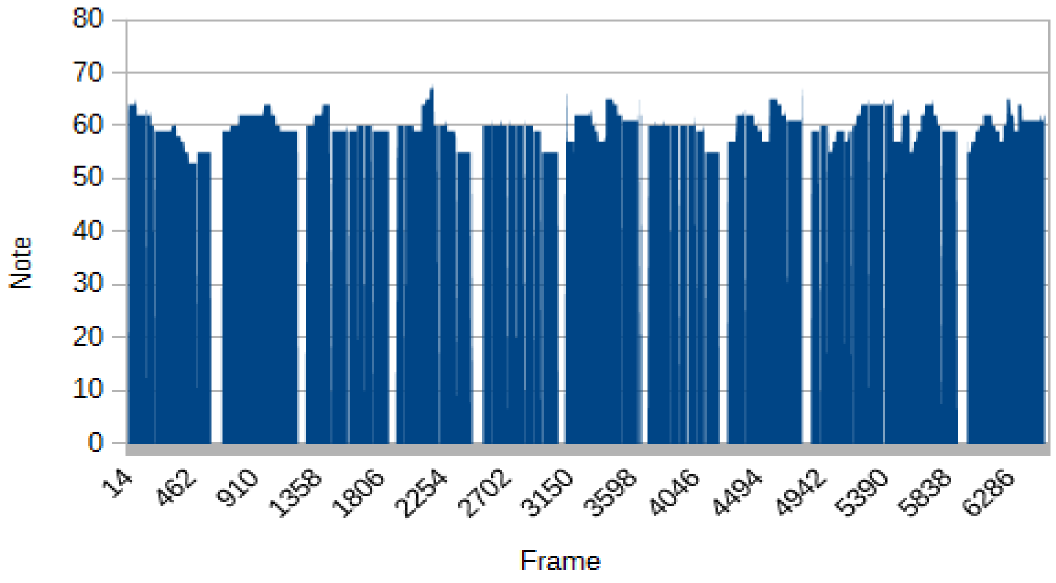


Figure 03.3 The notes for saxophone in the training set.

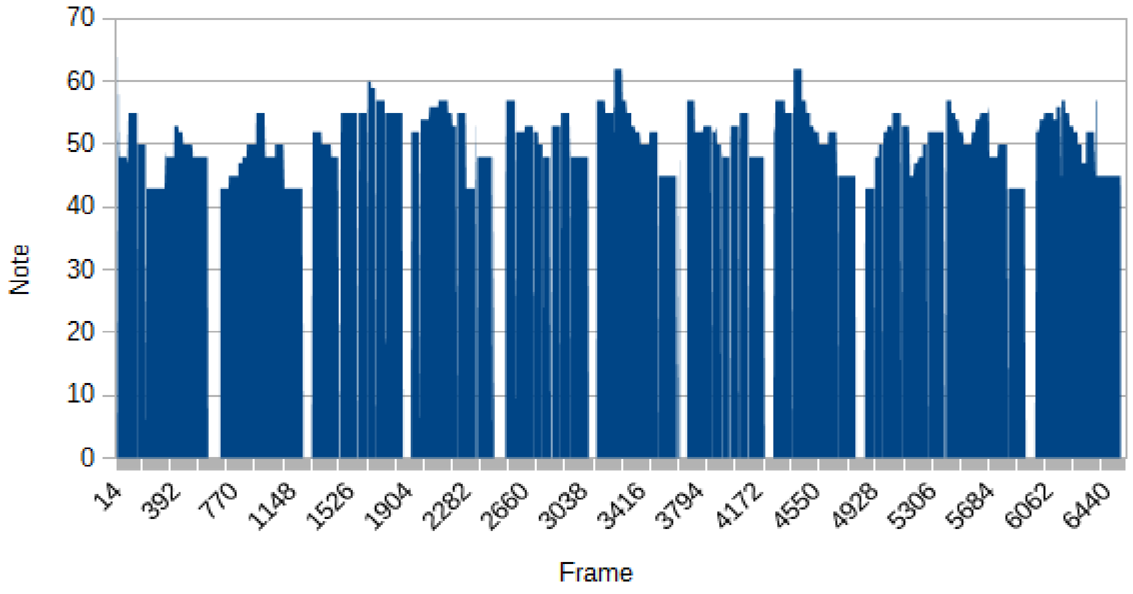


Figure 3.4 The notes for bassoon in the training set.

Data processing

As described in Subchapter 3.1, we process the input data by extracting the FFT as a characteristic feature, using YAAFE [16]. Other features have been experimented with as well: the MFCC (Mel-frequencies cepstrum coefficients), the Mel Spectrum, the Spectral Rolloff and the Spectral Flux, but neither of them yielded a result as good as the Magnitude Spectrum.

The values of the output have been rounded to the nearest integer to give exact notes. Since the number of notes did not correspond exactly to the number of frames, 5 notes (the difference between the two numbers was 5) were added in such a way that they were as spread out from each other as much as possible. These 5 notes took the values of the neighbouring notes.

After the feature extraction, it turned out that the feature vector had 1016 dimensions. The size of the dataset is 26228, which yields a ratio of approximately 26:1. Despite the high number the dimensions, the results were still good.

3.2.2 Evaluation metrics

To evaluate the performance of each of the algorithms on the data set, we used the following five metrics [18]:

- *Accuracy*
- *Precision*
- *Recall*
- *F_1-score*
- A metric based on the *confusion matrix – the confusion matrix score*, which is described further in this subchapter.

The values of all scores are represented as numbers between 0 and 1. In the following, we will briefly describe each type of metric.

The *accuracy* measures how many samples have been predicted correctly out of the total number that needed to be predicted. It is computed in the following way:

$$accuracy = \frac{number\ of\ samples\ predicted\ correctly}{total\ number\ of\ samples}$$

Precision and *recall* are best explained using the following diagram:

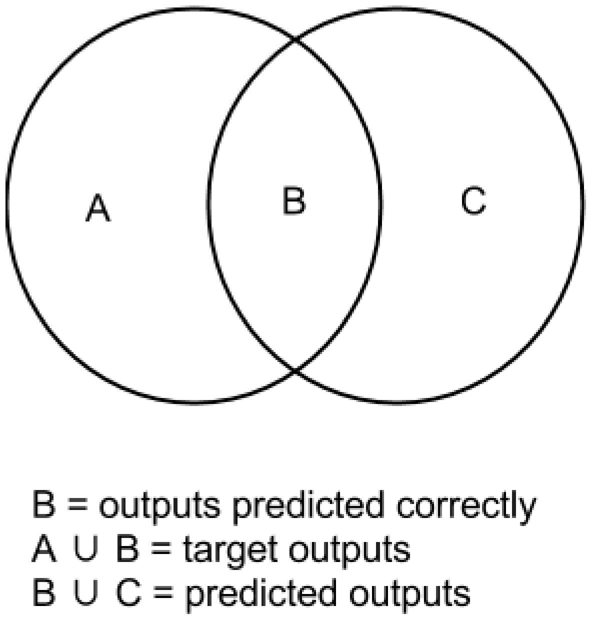


Figure 3.5 A diagram to explain precision and recall.

$$precision = \frac{|B|}{|B \cup C|}$$

$$recall = \frac{|B|}{|A \cup B|}.$$

Precision is a measure of the number of retrieved instances that are relevant, while recall measures how many relevant instances are correctly retrieved.

Precision and recall are specific to one class only. The result reported for precision and recall in our experiments represent the weighted average (where the weight is the number of samples of that class) of all the classes precision or recall. The F_1-Score is calculated as the harmonic mean of the precision and recall.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Another evaluation measure that we used is based on the confusion matrix. This metric shall be referred to as the *confusion matrix score*. This matrix shows on its diagonal the number of instances that were predicted correctly. The elements not on the diagonal represent the number of elements that were mislabelled. The bigger the diagonal values the better, because this means that there are that many correct predictions.

A curiosity regarding the performance of the application was to find out how many samples of each class were predicted correctly, using the confusion matrix. It was calculated how many samples of each class were predicted correctly by computing how much of each class is on the diagonal of the confusion matrix. The mean of those values was then evaluated (the confusion matrix score). The way this metric differs from the accuracy (which just states how many samples were predicted correctly) is by assigning the same importance to the prediction of *each* class.

3.2.3 Results

The algorithms chosen for experiments are Support Vector Classification (SVC) with an RBF kernel, K-Nearest Neighbours (K-NN) and Stochastic Gradient Descent Classifier (SGDClassifier). The experiments have been conducted with the help of the Scikit-learn library [14], a machine learning library written in Python. The sections below give a more detailed description of the experiments.

Support Vector Classification

The Scikit-Learn implementations of the SVC is a based on the LIBSVM library. A one-vs-one approach is used for support of multiclass classification. After literature research, the kernel function we chose was RBF (Radial Basis Function). It was also discovered that it is better if the weight of each class is 1. The other possibility would have been to set the weight of a class inversely proportional to the frequency of that class (the number of samples belonging to that class). The train accuracy reached **0.934** and the test accuracy was **0.753**. The confusion matrix score is **0.828**. Figure 3.6 shows the confusion matrix obtained for this experiment.

Table 3.2 presents the values of all used evaluation metrics, for this experiment. We notice that the average values for precision and recall are sufficiently high, thus indicating good classifications of the pitches.

Training score	Testing score	Confusion matrix score	Precision	Recall	F_1-Score
0.934	0.753	0.828	0.77	0.75	0.74

Table 3.2 Results of the SVC using the considered metrics.

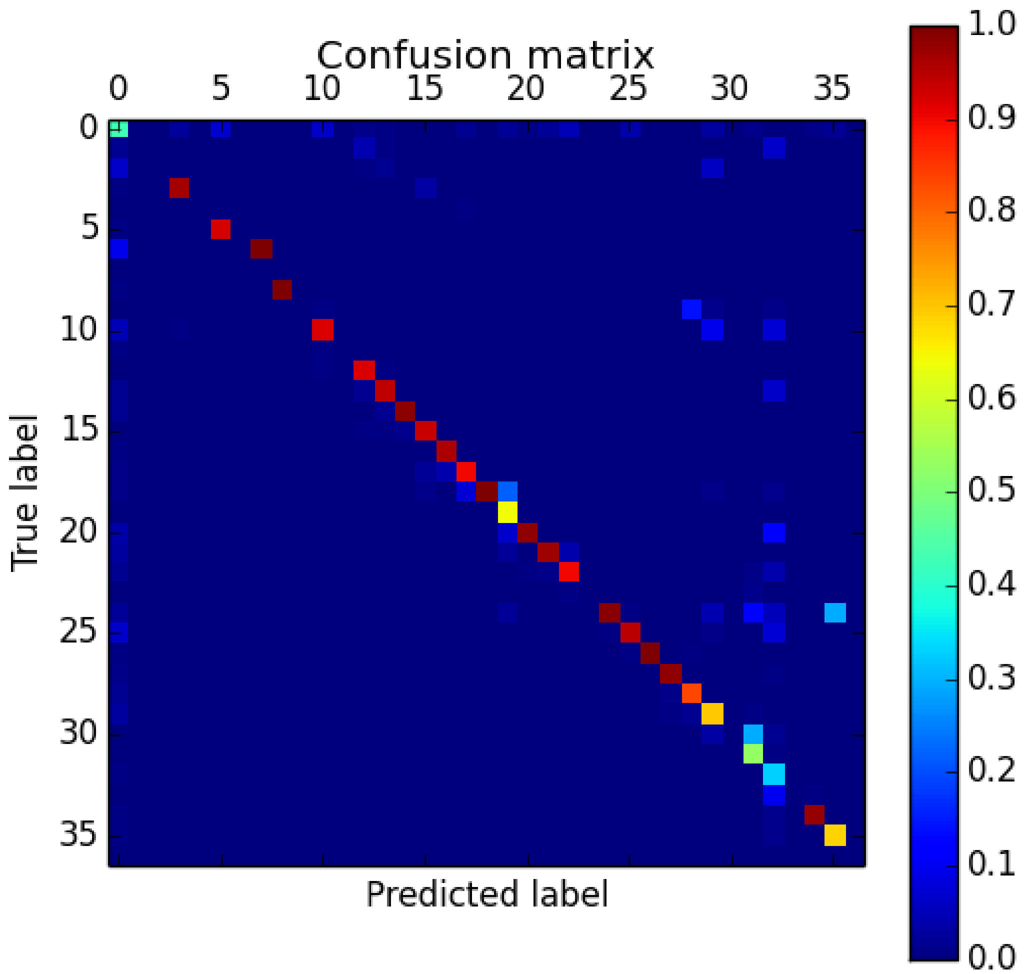


Figure 3.6 The confusion matrix for the SVC.

K-Nearest Neighbours

The Scikit-Learn algorithms available to compute the nearest neighbours are the Ball Tree, the KD Tree and the brute-force search. The parameter referring to the algorithm used was set to ‘auto’, which means that the K-NN algorithm will try to decide the most suited algorithm based on the dataset. After validation phase experiments, the parameter *k* was set to 5 (which is the default) and the weights with which the neighbours contributed were calculated according to the distance from them (the 1/d scheme was used, where d is the distance). The training accuracy is shown as **1.0** (even though there is no training phase per say, only storing of the data) and the test accuracy was **0.836**. The confusion matrix score is **0.684**. Figure 3.7 depicts the confusion matrix obtained for this experiment.

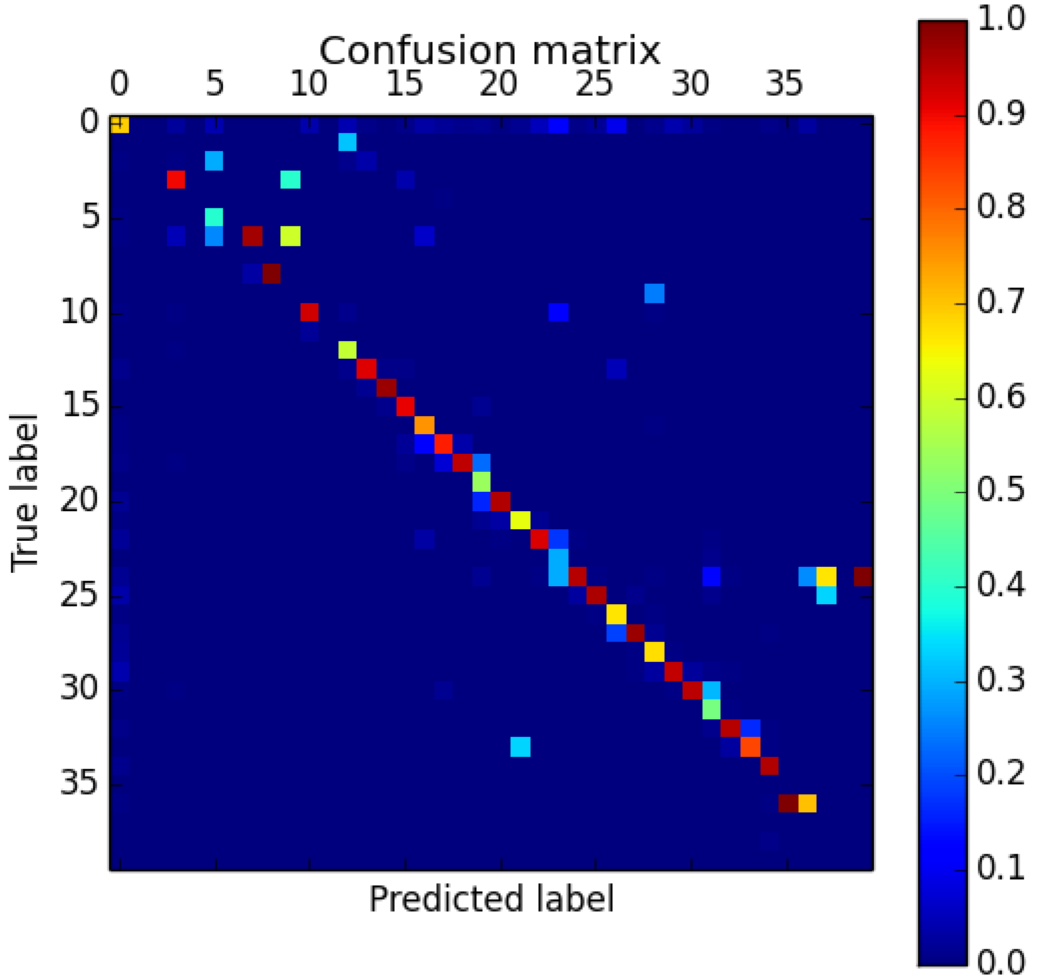


Figure 3.7 The confusion matrix for the K-NN.

Table 3.3 presents the values of all used evaluation metrics.

Training score	Testing score	Confusion matrix score	Precision	Recall	F_1-Score
1.0	0.836	0.684	0.82	0.84	0.82

Table 3.3 Results of the k-NN using the considered metrics.

Stochastic Gradient Descent Classifier

The SGDClassifier consists of a linear SVC with stochastic gradient descent training. It works in the following way: the gradient of the loss (cost function) is estimated each sample at a time and the learning rate decreases along the way. In the validation process, it was found that the number of iterations is a factor that affected the accuracy of the classifier, and the number chosen was 10. To avoid overfitting, the data was shuffled. Also in the case of the SGDClassifier, it was found out that the weights of each class should be one. The train accuracy reached **0.916** and the test accuracy was **0.832**. In figure 3.8 the depiction of the confusion matrix is shown. The confusion matrix score was: **0.769**. The reason this score is smaller than the accuracy score is because the estimator performs badly on certain classes. The most probable reason for this is that it doesn't have enough samples from those classes to train on.

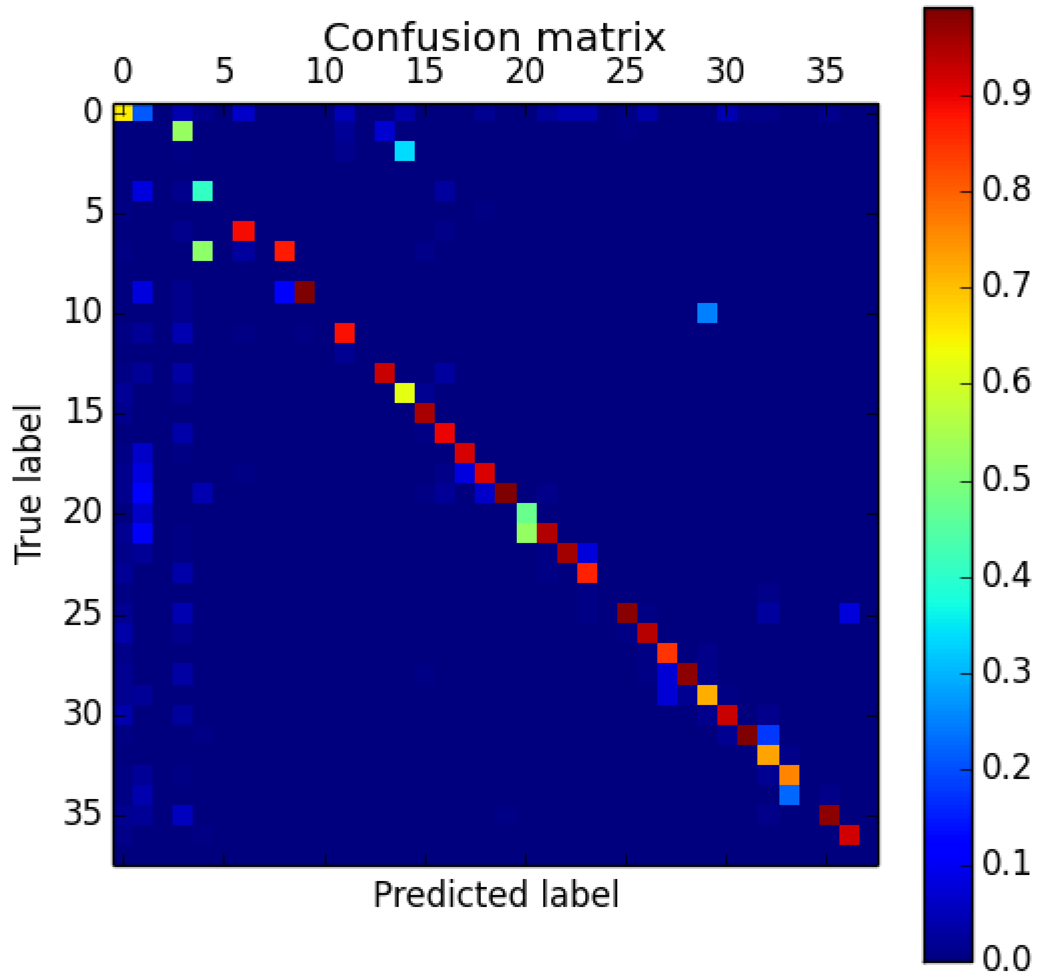


Figure 3.8 The confusion matrix for the SGDClassifier.

Table 3.4 shows the values of the considered evaluation metrics for this experiment.

Training score	Testing score	Confusion matrix score	Precision	Recall	F_1-Score
0.916	0.832	0.769	0.81	0.83	0.81

Table 3.4 Results of the SGDClassifier using the considered metrics.

Experiments have also been made with each instrument separately. Reducing the number of classes to just the ones specific for that type of instrument has also been tried, though the results did not change significantly.

3.2.4 Comparisons and discussions

Each of the algorithms experimented with were chosen for a reason. The K-NN was chosen because, despite its simplicity, has given great results in many classification applications. The SVC was chosen because it presented good results in past works in this field [7] [19]. The SGDClassifier was chosen because it is a very fast algorithm which allowed for multiple tests.

We compared the three algorithms from the point of view of each of the considered evaluation measures and also taking into consideration, separately, the training and testing scores. Figure 3.9 shows this comparison.

Concerning the training score, the SGDClassifier and the SVC performed very good, both of them obtaining values larger than 0.9. In the case of the k-NN, we must take into consideration the fact that all the k-NN algorithm does in the training phase is memorising the training set. It will always yield a value of 1. Concerning the testing, the performance of the k-NN was only slightly better than the one of the SGDClassifier (0.4%), but almost 10% higher than the performance obtained by the SVC. Regarding the confusion matrix score, the things are quite different, as the k-NN in this case obtains the lowest score. The confusion matrix score assigns the same importance to the prediction of each class, as opposed to accuracy which is computed in terms of total number of instances predicted correctly from all the classes. Therefore, although the accuracy was quite high for the k-NN, compared to the other two algorithms, the fact that the confusion matrix score is lower means that there are classes for which a big number of instances were correctly predicted, but there are also classes for which only a small number were properly classified.

As far as precision is concerned, the k-NN algorithm provided the best results (0.82), being 1% better than the SGDClassifier and 5% better than the SVC. This means that the k-NN had the highest fraction of retrieved samples that are relevant. In terms of recall, the k-NN is again the winner, having a recall value of 0.84, 1% better than the SGDClassifier and 9% better than the SVC. The recall measure evaluates the fraction of relevant samples that are retrieved.

The F1-score renders the harmonic mean of the precision and recall measures. The ranking remains the same: k-NN has the best value (0.82), being followed by the SGDClassifier that has a value of 0.81, and then by the SVC which holds a recall value of 0.74. However, when it comes to the confusion matrix score, the k-NN classifier's performance is poorer, meaning that it correctly predicted fewer samples for each class, when compared to the other two algorithms.

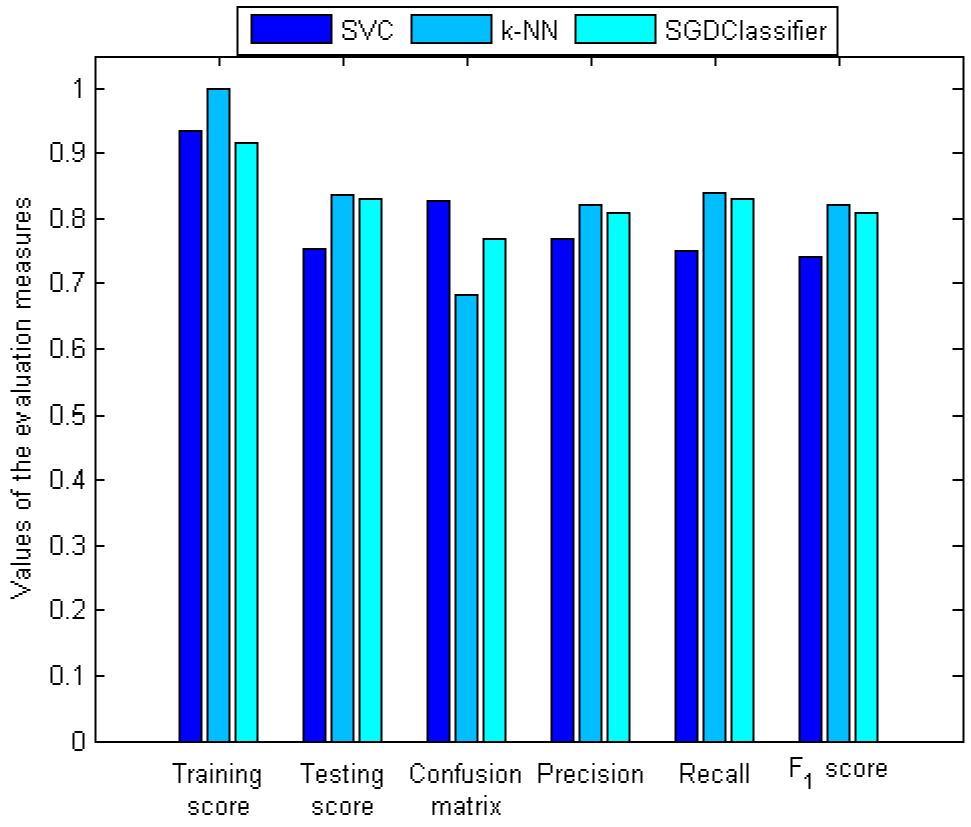


Figure 3.9 Comparisons of the three algorithms with regard to several evaluation measures.

As far as computation time is concerned, the SGDClassifier was by far the fastest, the SVC being the slowest. Table 3.5 shows the exact computational times for both training and testing. All the experiments presented in this section were carried out on a PC with an Intel Pentium Dual Core Processor with a frequency of 2.0 GHz and with 4 GB of RAM.

Algorithm	Training time (seconds)	Testing time (seconds)
SGDClassifier	26.9	0.21
K-NN	6.29	100.98
SVC	265.26	150.39

Table 3.5 The comparative computational times for training and testing.

3.3 Conclusions

This chapter presented the experiments and results of the three algorithms: Support Vector Machines, K-Nearest Neighbours and Stochastic Gradient Descent. For the experiments, a versatile music dataset [15] has been used, which is also presented. Several evaluation metrics have been used to assess the quality of the algorithms: the accuracy, the confusion matrix, precision, recall, F1-score and a metric based on the confusion matrix which shows how well the algorithms are able to predict each class. Each such metric is described in detail. The parameters used for the training of each such algorithm were also discussed, together with the reasoning behind choosing them.

Comparisons and discussions about the algorithms quality and computational time were also provided in the end of the chapter. The results offered by the considered evaluation measures were quite high for all three classification algorithms, this implying that all of them are suitable for pitch identification.

Chapter 4

Conclusions

In the present study we investigated the performances of several machine learning algorithms for the problem of musical pitch identification. Three techniques have been experimented on and compared, namely the K-Nearest Neighbour, the Support Vector Machine and the SGDClassifier.

The algorithms were tested on 4 chorales taken from a versatile music data set [15]. These inputs were processed and various features were extracted, the FFT proving to be the most relevant for music transcription. The tests showed that all three algorithms obtained good performances. Regarding the accuracy and the computational time, the most successful algorithm proved to be the SGDClassifier. Evaluating the quality of the application using precision, recall and f1-score measures, the k-NN algorithm was found to give the best results. The only evaluation metric for which the SVC won, was the confusion matrix score, a measure for the ability of the algorithm to correctly predict each class.

As future work, a good idea would be to make sure the number of samples for each class is equal (or approximately equal, at least). Since some estimators need a lot of training examples for each class, this could be the reason why some classifiers performed worse than others. Since the SVC doesn't perform so well on such a large number of classes, it could be an interesting study to create an SVC for each instrument (since each instrument has a certain interval of notes it can reach, which, of course, contains fewer notes than 128). Another interesting objective would be to experiment with other classifiers, such as Random Forrest Classifiers or different types of neural networks.

Bibliography

[1] T.M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[2] Piano Sonata No.11. Cantorion. [Online]. **http://cantorion.org/pieces/538/Piano-Sonata-No.-11**

[3] Manoj Desphande, "Musical Pitch Identification," 2011.

[4] A. Donkin, I. H. Witten G. Holmes, "WEKA: a machine learning workbench," 1994.

[5] M. Goadrich J. David, "The Relationship between Precision-Recall and ROC Curves," Madison, USA,.

[6] F.Y. Zhi R. Kidson, "Musical Note Identification, Violin Score Reproduction from Audio Recording," Garching bei Mnchen,.

[7] G.E. Poliner, *Classification-based Music Transcription*.: Columbia University, 2008, 2008, vol. 0549658041, 9780549658047.

[8] V. Vapnik C. Cortes, "Support-Vector Networks," vol. 20, no. 3, 1995.

[9] Cesar R. Souza. (2014, February) Kernel Functions for Machine Learning Applications. [Online]. **http://crsouza.blogspot.ro/2010/03/kernel-functions-for-machine-learning.html**

[10] A. Klatau R. Rifkin, "In Defense of One-Vs-All Classification," vol. 5, no. pages 101-104, 2004.

[11] B.V. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. Los Alamitos, CA, 1991.

[12] L. Bottou, "Stochastic Learning," in *Advanced Lectures on Machine Learning*. Ulrike: Springer Verlag, 2004.

[13] A. Ng, "Lecture Notes," 2008.

[14] G. Varoquaux, A. Gramfort, V. Michel et al. F. Pedregosa, "Scikit-learn: Machine Learning in Python," vol. 12, no. pages 2825-2830, 2011.

[15] B. Pardo, C. Zhang Z. Duan, "Multiple fundamental frequency estimation by modeling

spectral peaks and non-peak regions," vol. 18, no. 8, pages 2121-2133, 2010.

[16] S. Essid, T. Fillon, J. Prado, G. Richard B. Mathieu, "YAAFE, an Easy to Use and Efficient Audio Feature Extraction Software," 2010.

[17] LDS Ltd. (2003) Understanding FFT Windows. [Online]. **http://www.physik.uni-wuerzburg.de/~praktiku/Anleitung/Fremde/ANO14.pdf**

[18] A.C. Lorena, A.C. Carvalho, A.A. Freitas E.P. Costa, "A Review of Performance Evaluation Measures for Hierarchical Classifiers," 2007.

[19] A. Petrusa Ibanez, "Computationally efficient methods for polyphonic music transcription," Alicante, 2010.

[20] Chris Wittmann. (2005, Oct.) Tonalsoft - MIDI Note Number, Frequency Table. [Online]. **http://www.tonalsoft.com/pub/news/pitch-bend.aspx**