Determining Confidence Measures on Fundamental Frequency Estimations

Boyuan Deng

August 2016

Supervised by:
Denis Jouvet, Inria Nancy-Grand Est, France
Ingmar Steiner, Saarland University, Germany

Co-supervised by:
Yves Laprie, CNRS, France

Work done at Inria Nancy-Grand Est with the Inria MULTISPEECH team.

Submitted to Saarland University, University of Lorraine and Erasmus Mundus LCT Consortium.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Declaration

I hereby confirm that the thesis presented here is my own work, with all assistance acknowledged.

Nancy, France

Date: 09.08.2016   Signature

Acknowledgements

Abstract

Fundamental frequency estimation acts as the basis of many applications related to speech or music, such as speaker recognition, development of speech synthesis voices and music information retrieval. Various algorithms working in the time or frequency domain have been developed in the past and work quite well on good-quality signals. They become more error prone on lower-quality signals but none of them indicate how reliable the estimation results are, which is a crucial metric if the overall system makes decisions based on probabilities or multiple estimation algorithms are combined using ensemble methods.

In our work, we tackle this problem by developing and evaluating confidence measures based on machine learning approaches for several existing fundamental frequency estimation algorithms. Such confidence measure can be seen as the probability of the estimation result to be correct. Sophisticated neural network architectures, including long short-term memory networks (LSTM), are chosen for the construction of these measures. And numerous features computed on the signals and generated during estimations are used as input.

Besides confidence measures, our work also illustrates some empirical analyses on the robustness of existing fundamental frequency estimation algorithms. Data from an audio corpus is artificially distorted in a quantitatively controlled way, by modifying signal levels or adding certain types of noise at given signal-to-noise ratios, etc. Changes in error rates as well as the different error types (deviation or error in voiced/unvoiced decisions) are visualized and the underlying causes are analyzed in detail. The conclusions drawn may help guide the choice of fundamental frequency estimation algorithms depending on situations and needs.

Contents

2

List of Figures

3

List of Tables

Chapter 1

Introduction

The two keywords of our work are "fundamental frequency" and "confidence measure".

Fundamental frequency, defined as the lowest frequency of a periodic waveform, is a vital measure in audio processing. It can reflect a speaker's gender, emotional state, etc. in speech processing. And in any kind of content-based music processing, such as genre analysis, music retrieval and recommendation, etc. fundamental frequency is the bridge between audio and musical note representations. Applications in these fields explicitly conduct fundamental frequency estimations in early stages or produce intermediate representations closely related to fundamental frequency, especially by using neural network methods.

Real-life audio signals won't be perfectly periodic. Taking this into account, various fundamental frequency estimation algorithms working in the time domain, frequency domain, or both have been developed over the past decades. There is no single ideal estimation algorithm until now because of the variety of aperiodicity.

The original literature of existing fundamental frequency estimation algorithms usually reports average accuracies on clean signals, i.e. naturally generated sounds recorded by professional equipment. The error rates must be rising when the clean signals are distorted, but to what extent? Will there be differences between various algorithms' performance and what are the causes? We conduct a series of empirical analyses in chapter 4 to uncover these.

Then we move attention from average accuracies on a large audio corpus to the correctness of individual results. This is formalized as confidence measure, a concept widely used in statistics and computer science.

As far as we know, none of the existing fundamental frequency estimation algorithms provide confidence measures along with estimation results. It will be a meaningful piece of work to construct such confidence measures,

after which better results can be obtained by combining existing estimation algorithms using ensemble methods. They also enable us to integrate the algorithms into bigger systems.

We start from an important theorem linking confidence measures and neural network classifiers, then develop and evaluate confidence measures based on sophisticated neural network methods, which is shown in the latter part of this thesis.

Chapter 2

Background and Related Works

2.1 Fundamental frequency estimation algorithms

We focus on three widely used fundamental frequency estimation algorithms, namely Martin's algorithm (Martin 1982), SWIPE (Camacho 2007) and YIN (De Cheveigné and Kawahara 2002). They are implemented in the `JSnoori` toolkit[1].

Fundamental frequency estimations are conducted "locally" at given time offsets using the information within a very short context window. F0 values change rapidly in audio files (most obvious for music) so the results are valid locally only, too.

The implementations iterate through audio files, output estimated fundamental frequencies at time indices where the user has interest (usually defined through starting point and interval). The results can be visualized by software programs such as Wavesurfer[2] (Sjölander and Beskow 2000).

2.1.1 Common problems in fundamental frequency estimation

- Results doubling or halving with respect to correct values are not infrequently seen in immature autocorrelation-based methods.

- Voiced/unvoiced decision is another common source of errors.

2.1.2 Martin's algorithm

Martin's algorithm is a spectral method which detects fundamental frequency by maximizing the intercorrelation between the power spectrum and a spectral comb.

---

[1] `http://jsnoori.loria.fr/`
[2] `http://www.speech.kth.se/wavesurfer/`
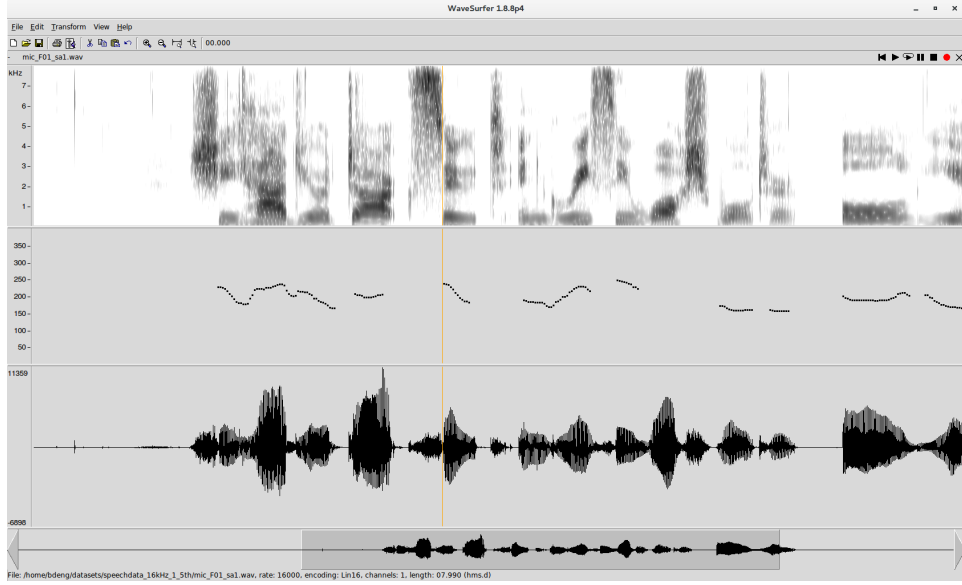
Figure 2.1: Spectrogram, plot of reference F0 values, and waveform shown in Wavesurfer.

On the frequency scale, the comb function is a train of teeth of decreasing amplitude. The teeth are regularly spaced but the uniform interval ω_p itself can be varied.

$$C(\omega_p, \omega) = \sum_n n^{-\frac{1}{k}} \delta(n\omega_p - \omega) \tag{2.1}$$

n is teeth order. δ is the Dirac delta function.

The signal power spectrum $|F(\omega)|$ is reduced to parabolas centered on the spectral peaks, becoming $|F'(\omega)|$.

The correlation between the modified power spectrum and the comb function is then

$$\begin{aligned}
I(\omega_p) &= \int_0^\infty C(\omega_p, \omega) |F'(\omega)| \, \mathrm{d}\omega \\
&= \sum_n n^{-\frac{1}{k}} |F'(\omega_p)|
\end{aligned} \tag{2.2}$$

It reaches maximum when the fundamental frequency of the comb ω_p is equal to the fundamental frequency ω_0 in the power spectrum.

Martin's algorithm is a simple yet effective algorithm. The `JSnoori` toolkit implements an enhanced version of Martin's algorithm which additionally takes the energy histogram and voicing jumps into account, etc.

9
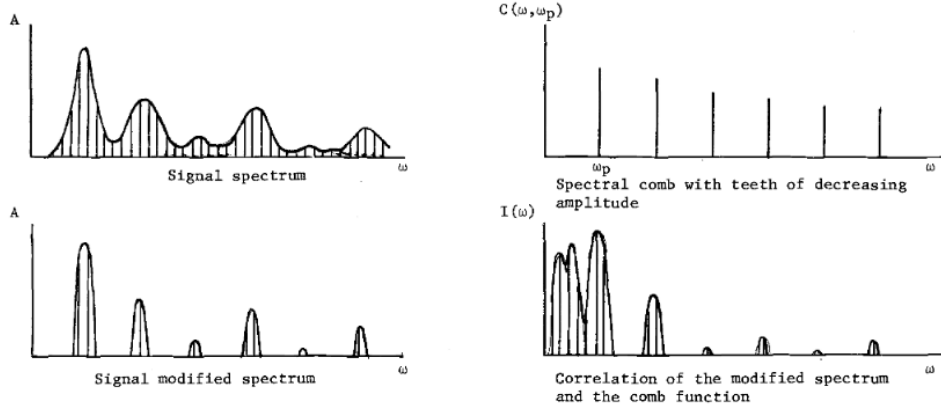
Figure 2.2: Steps of Martin's algorithm (from (Martin 1982)).

2.1.3 SWIPE

The core idea of Sawtooth Waveform Inspired Pitch Estimator (SWIPE) is that if a signal is periodic with fundamental frequency f, its spectrum must contain peaks at multiples of f and valleys in between.

Since each peak is surrounded by two valleys, the average peak-to-valley distance (APVD) for the k-th peak is defined as

$$d_k(f) = \frac{1}{2}\left[\left|X(kf)\right| - \left|X\left(\left(k - \frac{1}{2}\right)f\right)\right|\right] + \frac{1}{2}\left[\left|X(kf)\right| - \left|X\left(\left(k + \frac{1}{2}\right)f\right)\right|\right]$$

$$= \left|X(kf)\right| - \frac{1}{2}\left[\left|X\left(\left(k - \frac{1}{2}\right)f\right)\right| + \left|X\left(\left(k + \frac{1}{2}\right)f\right)\right|\right]$$

$$\tag{2.3}$$

X is the estimated spectrum of the signal which takes frequency as input and outputs corresponding density.

By averaging over the first n peaks, we obtain the global APVD, which is to be maximized.

$$D_n(f) = \frac{1}{n}\sum_{k=1}^{n} d_k(f)$$

$$= \frac{1}{n}\left[\frac{1}{2}\left|X\left(\frac{f}{2}\right)\right| - \frac{1}{2}\left|X\left(\left(n + \frac{1}{2}\right)f\right)\right| + \sum_{k=1}^{n}\left(\left|X(kf)\right| - \left|X\left(\left(k - \frac{1}{2}\right)f\right)\right|\right)\right]$$

$$\tag{2.4}$$

There are other improvements such as blurring the harmonics, warping the spectrum and weighting the harmonics. Please refer to the cited long report (Camacho 2007) for all the details.

2.1.4   YIN

YIN is basically a correlation-like method working in the time domain.

A traditional autocorrelation method chooses the highest non-zero-lag peak in the autocorrelation function (ACF, Equation 2.5) by exhaustive search within a range of lags.

$$r_t(\tau) = \sum_{j=t+1}^{t+W} x_j x_{j+\tau} \tag{2.5}$$

x_t is the value of the discrete signal at time index t. $r_t(\tau)$ is the autocorrelation function of lag τ calculated at time index t, and W is the integration window size.

It requires elaborated correction algorithms to work well, as the choice of range is vital. If the actual period lies outside of the chosen range, double or half the correct value is obtained.

YIN instead finds the period by minimizing the difference function $d_t(\tau)$, which is much less sensitive to amplitude changes while ACF peak amplitudes grow with lag rather than remaining constant when the signal amplitude increases with time.

$$d_t(\tau) = \sum_{j=1}^{W} (x_j - x_{j+\tau})^2 \tag{2.6}$$

By changing the objective function into the cumulative mean normalized difference function (Equation 2.7), YIN is free from the influence of a strong resonance at the first formant (F1) and eliminates the upper frequency limit of the search range.

$$d_t'(\tau) = \begin{cases} 1, \text{if } \tau = 0, \\ \dfrac{d_t(\tau)}{\dfrac{1}{\tau}\sum_{j=1}^{\tau} d_t(j)}, \text{otherwise} \end{cases} \tag{2.7}$$

Absolute threshold, parabolic interpolation and best local estimate further improve YIN's performance.

2.2 Neural networks

2.2.1 Introduction to neural networks

Mathematically speaking, (artificial) neural networks are multiple levels of non-linear operations. By adding more layers and more units within a layer, a deep network can represent functions of increasing complexity.

Most tasks that consist of mapping an input vector to an output vector can be accomplished via relatively deep neural networks, given sufficiently large models and sufficiently large datasets of labeled training samples.

Neural networks (or more recently, "deep learning") form a rich and diverse field of research. A complete description is beyond our scope. In the following we will introduce the neural network methods involved in our work.

2.2.2 Multilayer perceptron

Multilayer perceptrons, also called feedforward neural networks, are basic but essential models in this field. They have one or more hidden layers between input and output layer.
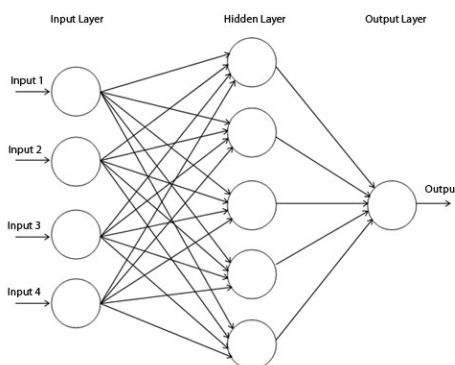
Figure 2.3: An example MLP network with one hidden layer.

"Feedforward" means that information flows in one direction from input to output layer (forward).

Every node calculates the weighted sum of its inputs $\mathbf{x}$ and apply some activation function f to get the output y, $y = f(\mathbf{w}^\mathsf{T}\mathbf{x} + b)$.

Rectified linear unit (ReLU) (Nair and Hinton 2010), defined as $f(x) = \max(0, x)$, is the optimal choice for activation function inside MLP nowadays.

It has several advantages over tanh, etc.

- piecewise polynomial non-linearity

The Rectified Linear Activation Function
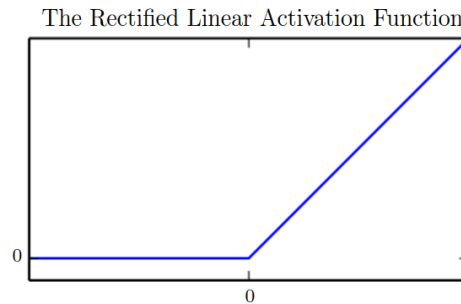
Figure 2.4: Rectified linear unit (ReLU).

- no vanishing gradient problem or exploding effect

- sparse activation

2.2.3 Recurrent networks and long short-term memory

There are no feedback connections in multilayer perceptrons by which outputs of the model are fed back into itself. In order to remember information over time, recurrent connections must be adopted.

Recurrent neural network (RNN) is a popular choice for processing sequential data. Its chain-like structure is able to capture temporal dependencies.
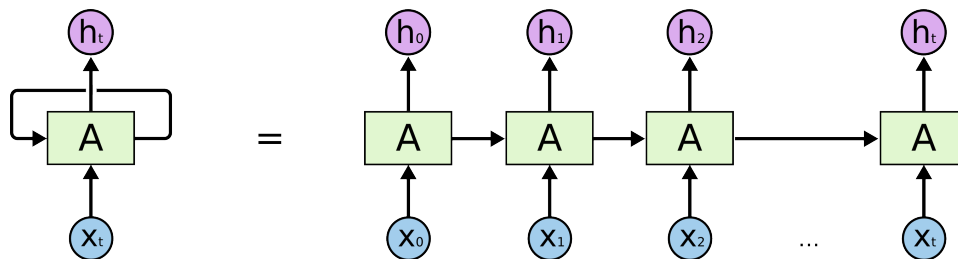
Figure 2.5: Unrolling a recurrent node.

The structures of recurrent neural networks are rather flexible, depending on the task and especially on which operations should be conducted on sequences.

- one to many: tasks such as generating captions from images.

- many to one: sequence classification, including sentiment analysis and **our task**.

- many to many: neural machine translation, neural network TTS, etc.

---

[3]`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`
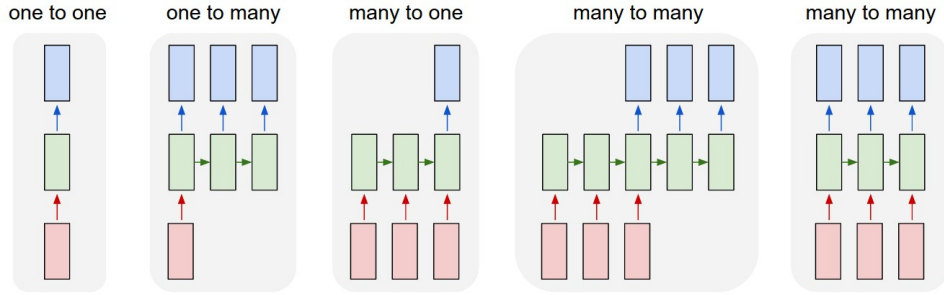
Figure 2.6: Various recurrent structures compared with MLP (leftmost). Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state [3].

Bengio (Bengio, Simard, and Frasconi 1994) shows that, though theoretically possible, in practice it's very hard to tune vanilla RNNs (hidden output directly used in the next time step) to handle long-term dependencies.

Long short-term memory network (LSTM) is a variant of RNN explicitly designed to tackle the long-term dependency problem.
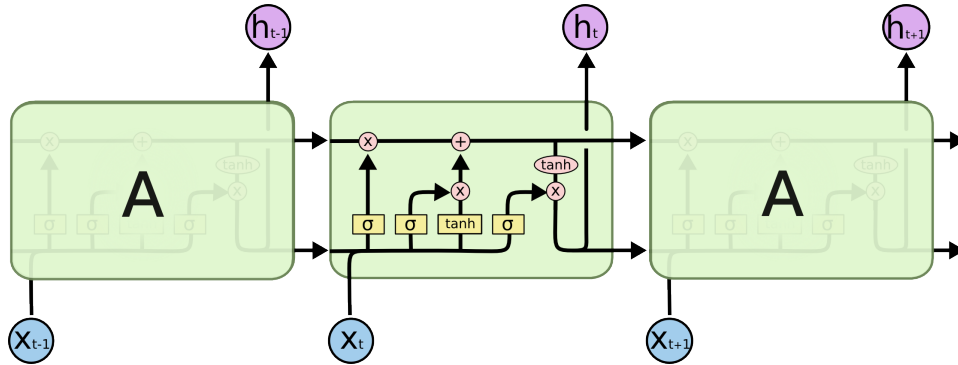
Figure 2.7: Details of an LSTM cell[4].

The cell state (horizontal line running through the top of the diagram) enables information to flow into the next time step freely. Any modification to the cell state is regulated by "gates", composed of a `sigmoid` (σ) layer and a point-wise operation. From left to right are "forget gate", "input gate" and "output gate", respectively.

Forget gate decides the proportion f_t of previous cell state to keep, individually on each dimension.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

---

[4] `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Input gate produces new candidate values $\tilde{C}_t$ and an input proportion i_t.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Then cell state is updated to C_t and new hidden output h_t is produced, regulated by output gate.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

"$\cdot$" is the dot product of two vectors and "$*$" is multiplying by scalar element-wise.

2.2.4 Training of neural networks

Neural networks are usually trained using gradient-based optimization methods, to minimize the cost function (also called loss function). A cost function measures the quality of a particular set of parameters based on how well the induced scores agree with ground truth labels.

In classification settings, the typical cost function used is cross entropy,

$$H(p, q) = - \sum_{x \in X} p(x) \log q(x), \tag{2.8}$$

where p is the ground truth probability distribution of labels and q is the distribution output by the neural network.

Parameters are updated by backpropagation, which is a way of computing gradients of expressions through recursive application of chain rule.

Modern neural networks often utilize dropout (Srivastava et al. 2014) to prevent overfitting: during training, one or more nodes are randomly "switched off" temporarily. Their weights won't be updated during that period, nor will they affect the learning of other nodes. They will be "switched on" again after some number of iterations. This is equal to learning an exponential number of "thinned" networks and averaging their predictions at testing time.

2.3 Confidence measures

While accuracy or average error rate have been used as standard metrics to characterize the performance of various systems, it is always useful to have a sense of correctness of each result.

Mathematically speaking, it is

the probability of the result to be correct

or

the true posterior probability $P(c_i = 1|X)$

in Bayesian approach. $c_i = 1$ means the class label is c_i for sample X.

2.3.1 Confidence measures in speech recognition

In speech processing, confidence measures have been investigated since a long time ago and are commonly used in the field of automatic speech recognition (ASR).

Any applicable real-world ASR system must be able to distinguish error prone results from its candidate outputs. This requires the ASR systems to automatically assess reliability for every decision made by the systems. Researchers propose to calculate a score between 0 and 1, i.e. confidence measure, to quantify this reliability.

A confidence measure can be calculated on every recognized word, indicating how likely it is to match the audio segment, or on the whole result corresponding to the input utterance. It is vital for achieving *robust speech recognition* and is still a trending topic.

2.3.2 Approaches to confidence measures

There are roughly three major categories of methods for computing confidence measures in ASR (Jiang 2005).

Combination of predictor features

Any feature can be called a predictor if its probabilistic distribution of correctly recognized words is clearly distinct from that of misrecognized words. The predictor features may have to be collected within the recognition process.

Some common predictor features include *N-best related scores*, *acoustic stability*, *phoneme or word durations*, *language model score* and *parsing results*.

Then all predictor features are combined in a certain way to generate a single score to indicate correctness of the recognition decision. This can

be done by various popular machine learning algorithms, but usually the combination methods cannot significantly improve over the best predictor feature.

Hypothesis testing

The confidence measure is formulated as a statistical hypothesis testing problem, under the framework of utterance verification.

For a given speech segment X, assume that an ASR system recognizes it as word W which is represented by a model λ_W. Then two complementary hypotheses are proposed, namely null hypothesis H_0 and the alternative hypothesis H_1.

- H_0: X is correctly recognized and truly comes from model λ_w.

- H_1: X is wrongly classified and is not from model λ_w.

Then test H_0 against H_1 to determine whether to accept the recognition result or reject it. The final solution ends up with calculating and evaluating likelihood ratio or Bayes factors.

As posterior probability

Conventional ASR systems find the most likely sequence of words $\hat{W}$ for acoustic observation X by *maximum a posterior* decision rule.

$$
\begin{aligned}
\hat{W} &= \arg\max_{W \in \Sigma} p(W|X) \\
&= \arg\max_{W \in \Sigma} \frac{p(X|W)p(W)}{p(X)} \\
&= \arg\max_{W \in \Sigma} p(X|W)p(W)
\end{aligned}
\tag{2.9}
$$

Σ is the set of all permissible sentences.

In theory, the posterior probability $p(W|X)$ is a good confidence measure.

However, most practical ASR systems ignore $p(X)$ during decision making so raw ASR scores are inadequate. And actually it is impossible to margin over all combinations of words, phonemes, noises, and other events to get $p(X)$.

Therefore, approximate methods are adopted. Usually, one word lattice or graph is generated by the ASR decoder for every utterance. Then the posterior probability of each recognized word or the entire hypothesized sentence can be calculated based on the word-graph from an additional post-processing stage.

2.3.3   Confidence measures and neural networks

As shown above, it is a complicated task to calculate the posterior probabilities "online" during speech recognition.

However, posterior probabilities may also be estimated "offline", under the framework of classification, when we have ground truth telling us whether the results are correct.

Theorem. *Bayesian probabilities are estimated when desired network outputs are 1 of M (one output unity, all others zero) and a squared-error or cross-entropy cost function is used. (Richard and Lippmann 1991)*

- Neural network binary classifiers having only one node with `sigmoid` activation function in the output layer, are able to output a value between 0 and 1 for classification decision. The higher the value, the more it favors the positive class.

- For multiclass classification problems, usually `softmax` activation function is used in the output layer. The class with the highest activation value (also between 0 and 1) gets output.

According to the theorem proved in (Richard and Lippmann 1991), **the above activation values estimate Bayesian a posteriori probabilities, acting directly as confidence measures**, as long as we choose squared-error or cross-entropy as cost function.

Under such a framework, multiclass neural network classifiers are applicable for estimating posterior probabilities for different candidate words.

And in our work about fundamental frequency estimation, it's a binary classification problem between "estimation is correct" and "estimation is incorrect". What we want is the posterior probability for "estimation is correct".

Chapter 3

Corpora

3.1 Speech corpus

The Pitch Tracking Database from Graz University of Technology (PTDB-TUG) (Pirker et al. 2011) is used throughout the remainder of this thesis.

This dataset contains microphone and laryngograph signals of 20 English native speakers reading out sentences from the TIMIT corpus (Garofolo et al. 1993). Both microphone signals and laryngograph signals were recorded at 48 kHz sampling rate, 16 bit resolution, with the type of encoding signed PCM.

Additionally, reference pitch trajectories extracted on the laryngograph signals using the RAPT algorithm (Talkin 1995) are provided. In total it has 4720 recorded utterances.

It is the first audio corpus that meets the following requirements:

- a substantial amount of speech data composed of phonetically rich sentences that allows for meaningful training of speaker-dependent models

- a variety of female and male speakers such that a multi-pitch tracker can be evaluated seriously

The latter feature is crucial to our work since frequency ranges of female and male voices differ a lot. The gender distribution is 50:50, and age varies from 22 to 48 years as described in the specifications.

In the corpus, audio files are encoded in `WAV` format and the filename contains signal type, speaker ID and sentence ID. Only microphone signals are used for testing. For any microphone signal file `mic_*.wav`, the corresponding file containing reference values is `ref_*.f0`, in CSV-like format.

According to the default pitch extraction script and the Snack Sound Toolkit[1] manual, indices of the reference values start at 16 ms and have a uniform interval of 10 ms until the end.

3.2   Noise corpus

The PTDB-TUG corpus contains only clean signals recorded inside a recording studio. Additional noise audios are needed in latter steps of our work. We pick five noise audio files from the NOISEX-92 corpus (Varga and Steeneken 1993), which is often used in the field of automatic speech recognition.

- `babble.wav`: 100 people speaking in a canteen and individual voices are slightly audible.

- `factory1.wav`: recorded near plate-cutting and electrical welding equipment in a factory.

- `factory2.wav`: recorded in a car production hall.

- `pink.wav`: acquired by sampling high-quality analog noise generator (Wandel & Goltermann). Exhibits equal energy per 1/3 octave.

- `white.wav`: acquired by sampling high-quality analog noise generator (Wandel & Goltermann). Exhibits equal energy per Hz bandwidth.

They have the same duration of 235 s, resolution of 16 bit and sampling rate at 19.98 kHz.

3.3   A unified sampling rate

`JSnoori` requires that input audio files are sampled at 16 kHz. We use `SoX`[2] to conduct the conversion.

---

[1] `http://www.speech.kth.se/snack/`
[2] `http://sox.sourceforge.net/`

Chapter 4

Empirical Analyses of the Algorithms

We conduct a series of empirical analyses of performance of the fundamental frequency estimation algorithms. This is done by altering the audio files in various ways, running the algorithms and comparing the results with reference values to calculate error rates.

Estimation results and reference values are technically time series at (usually) uniform intervals along the time axis of signals. They don't necessarily have the same starting point or interval. So we interpolate estimation results to obtain matching values at reference indices. In order to maintain the clear boundaries between voiced and unvoiced frames and avoid introducing erroneous interpolated intermediate values, and since the intervals are very short even in voiced areas, **nearest-neighbor interpolation** is applied throughout.

(a) Linear interpolation
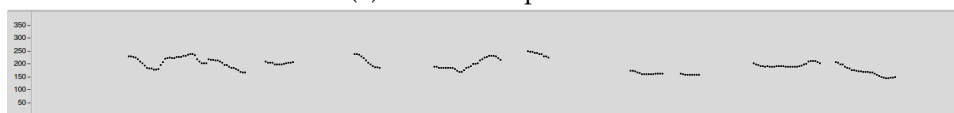
(b) Actual values

Figure 4.1: Wavesurfer is able to plot time series with connecting lines, analogous to linear interpolation. **The actual fundamental frequency values within steep areas are either zero or the neighboring nonzero value, not on the connecting lines.**

We set the maximum deviation percentage from reference values to be

20 %, within this range the estimation results are considered correct.

The different experiments we conduct are as follows. Elaborated figures and statistics are provided.

4.1 Apply offsets to indices

Fundamental frequency estimation algorithms usually compute features over a sliding window and use the centers of the windows as time indices for estimated values. This is sometimes inaccurate, as the proper indices may deviate from the centers, and the specific offsets depend on the internal mechanisms and implementations of the algorithms, which are not always explicitly documented.

We use grid search to empirically find the optimal offset for every algorithm implemented in `JSnoori`: before interpolating the estimation results, pick an offset from some range and apply it to all the indices. The optimal offset will minimize the error rate.
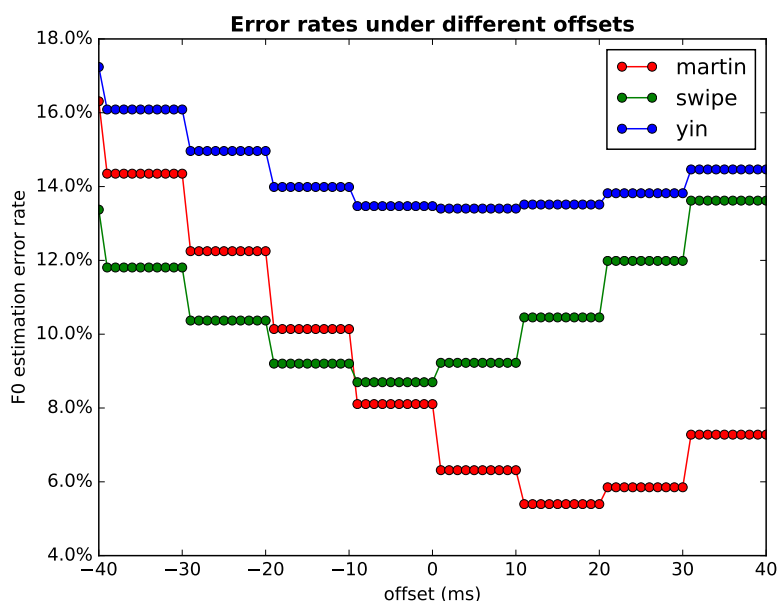
Figure 4.2: F0 estimation error rates under different offsets.

Figure 4.2 shows the error rates when different offsets are applied. There are plateaus in the curves which means that nearest-neighbor interpolation works. The optimal offsets we find are shown in Table 4.1.

We apply these optimal offsets throughout the following experiments.

Algorithm	Martin's	SWIPE	YIN
Optimal offset (ms)	16	−4	6

Table 4.1: Optimal offsets for the three algorithms.

4.2   Modify signal levels

Because of the 16-bit integer representation of the signal values in the WAV files, when the signal level is lowered, the precision decreases, and consequently, it is expected that the accuracy of the fundamental frequency estimation will decrease, hence this set of experiments aims at better understanding how the various fundamental frequency estimation algorithms behave when the signal level is modified (lowered).
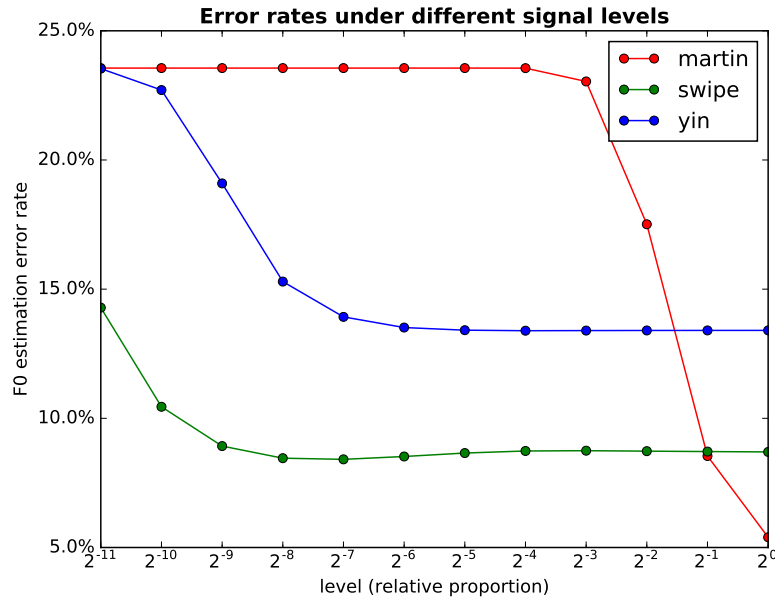
Figure 4.3: F0 estimation error rates under different signal levels.

We use the SoX utility to adjust the signals to different levels (scale the signal values). The resulting error rates are plotted in Figure 4.3.

Comment and Analysis

For all three algorithms, error rates converge to about 23.56 % when signal level approaches zero. This is the proportion of voiced samples (i.e. values > 0 Hz) in reference values. **When signal level is very low,**

> **the algorithms output 0 Hz for all data points, thus only the estimation results aligned to voiced reference values are wrong.**
>
> From the plot we can see that Martin's algorithm performs well when signal level is unmodified, but is very sensitive to further changes. In contrast, SWIPE and YIN's performance are rather steady and the results are always usable since real-life signal level changes are much smaller. (Also, SWIPE works better than YIN in this task.) **This is because that enhanced Martin's algorithm explicitly takes energy of the signal into account while SWIPE and YIN do not. Starting from 2^{-3}, enhanced Martin's algorithm decides that most of the frames are unvoiced, which is overpessimistic.**

4.3 Add noise by level

We then evaluate the effects of noise. `SoX` is used again to add noise types to the original signals.

This is done by scaling the signal level of a noise file and then mixing it with the original audio file. To avoid clipping, we calculate the sum of peak amplitudes (maximum absolute value) of the two files before mixing. If it exceeds the largest value that can be represented (though clipping may not actually happen because the two maxima usually don't occur at the same time index), we apply another scaling on the two files.

$$Y = \alpha S + \alpha \lambda N, \quad \alpha = \begin{cases} 1 & , \mathrm{pa}(S) + \mathrm{pa}(\lambda N) < 1 \\ \frac{1}{\mathrm{pa}(S) + \mathrm{pa}(\lambda N)}, & \mathrm{pa}(S) + \mathrm{pa}(\lambda N) \geq 1 \end{cases} \quad (4.1)$$

Y, S and N are output, speech, and noise signals, respectively. α and λ are scale factors. $\mathrm{pa}()$ is the function returning peak amplitude.

We first choose λ from {0, 0.0125, 0.025, 0.05, 0.1, 0.2, 0.4, 0.8} and apply the white noise generated by SoX, which always has the maximum possible peak amplitude 1. Results are shown in Figure 4.4.

Then we randomly sample noises from the five noise audio files in the NOISEX-92 corpus: `babble.wav`, `factory1.wav`, `factory2.wav`, `pink.wav` and `white.wav`. The last one is also white noise but with a different volume.

Results are shown in Figure 4.5 and Figure 4.6[1] These statistics objectively reflect the three algorithms' robustness to different kinds of noise at real-life signal levels.

Noise levels all correspond to λ, the five noise signals have very different signal levels or volumes.

---

[1]Legends indicate (`noise`, `algorithm`) combinations. The same applies to all figures of this type.
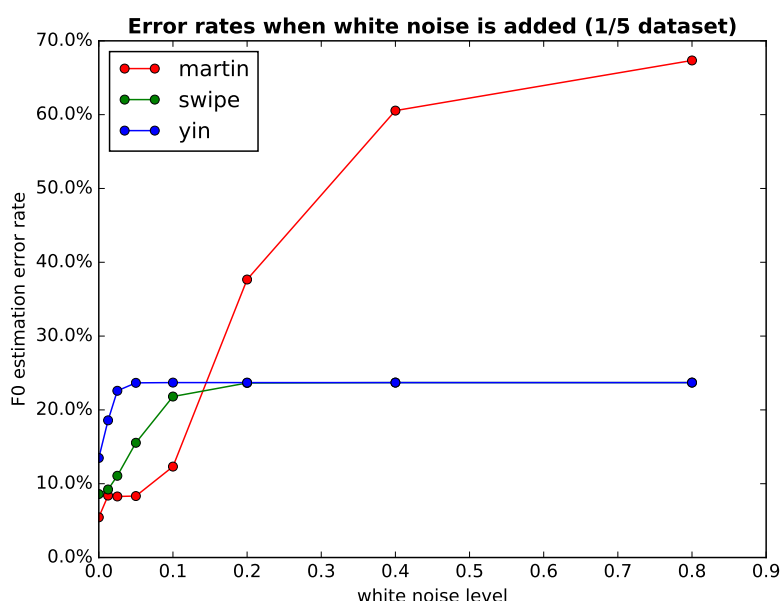
Figure 4.4: F0 estimation error rates when white noise is added. White noise level corresponds to λ.

Comment and Analysis (Figure 4.4)

Martin's algorithm is more resistant to distortion when the noise level is relatively low, and has better performance than SWIPE and YIN. Though its error rate continues rising afterwards, SWIPE and YIN's error rates converge to the proportion of voiced samples again so the results are meaningless, too. **This kind of "blowup" is again caused by enhanced Martin's algorithm's decision mechanism based on energy. More and more noise saturates the signal and enhanced Martin's algorithm does a lot of useless estimations on originally unvoiced frames.**

YIN's performance drops very quickly. **It is basically correlation-like and therefore less robust to noise as the waveform changes drastically.**

Comment and Analysis (Figure 4.5 and Figure 4.6)

Figure 4.5 further verifies our first conclusion about Figure 4.4. The higher the volume of a noise signal, the more quickly the error rate of

25

enhanced Martin's algorithm rises. `white.wav` has much lower volume than the white noise generated by `SoX`, so the error rate even doesn't change much all the way.

In general, all three fundamental frequency estimation algorithms are more easily influenced by real-life noise than generated noise signals (`pink.wav` and `white.wav`) when the noises are of realistic volumes or "recorded as it is".

4.4   Add noise by signal-to-noise ratio

A tool named `FaNT - Filtering and Noise Adding Tool`[2] enables us to add a noise signal at a given signal-to-noise ratio (SNR), which makes the results (in Figure 4.7) easier to interpret and more numerically referable. Noise signals are first sampled across the five noise files. All audio files need to be converted to RAW format first, which is also done by `SoX`.

It can be seen from Figure 4.8 that when the audio is highly distorted, lots of details in spectrogram and waveform are lost and it's really hard to do fundamental frequency estimation. SWIPE and YIN tend to output $0\,\text{Hz}$ for more and more frames when the signal-to-noise ratio is gradually reduced.

We further distinguish between error types (Figure 4.9), which helps us analyze the drawbacks of the algorithms under certain circumstances. Three types of errors are investigated, namely

- "deviated" (voiced both in reference and result but the result deviates too far from the reference)

- "incorrectly voiced" (unvoiced in reference but voiced in result)

- "incorrectly unvoiced" (voiced in reference but unvoiced in result)

When the signal-to-noise ratio is fixed across different types of noise, it will be easier to investigate the effect of a noise signal's acoustic properties on fundamental frequency estimation.

Next, we sample random noise from the five files separately, "decomposed" Figure 4.11 and Figure 4.12 are provided.

Comment and Analysis (Figure 4.7 and Figure 4.9)

SWIPE and YIN have virtually only "incorrectly unvoiced" errors. When the audio is highly distorted, SWIPE and YIN consider all frames to be unvoiced, while enhanced Martin's algorithm tends to do the opposite and introduce a lot of "incorrectly voiced" errors. At the same

---

[2] http://dnt.kr.hs-niederrhein.de/index964b.html

Figure 4.5: F0 estimation error rates for each algorithm (Martin's at the top, SWIPE in the middle, YIN at the bottom) when random noise is sampled from different audio signals. Subfigures have different y-axis limits.

Figure 4.6: F0 estimation error rates of the various algorithms (Martin's represented by circles, SWIPE represented by squares, and YIN by diamonds) for each type of noise. Subfigures have different y-axis limits.
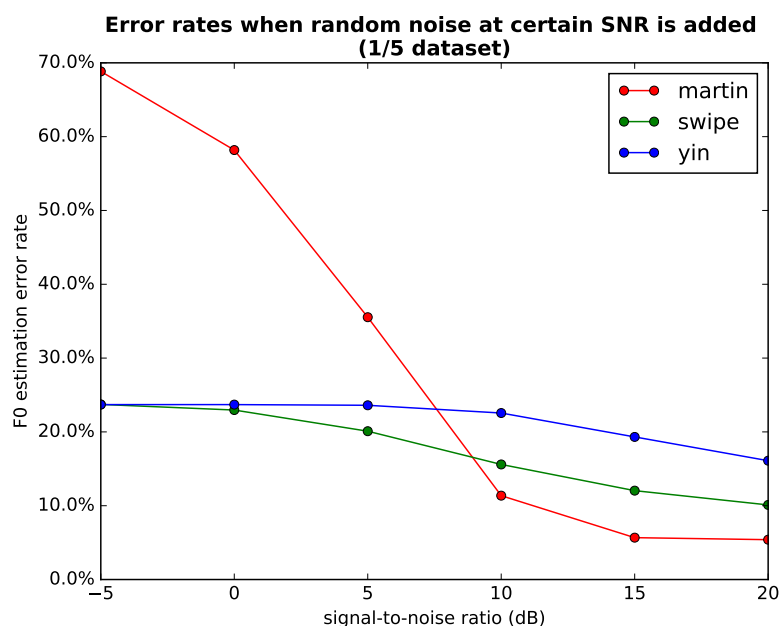
Figure 4.7: F0 estimation error rates when random noise at certain SNR is added.

time, estimation results on voiced frames are more or less acceptable.

This phenomenon indicates that a more sophisticated voiced or unvoiced decision mechanism can greatly improve all three algorithms.

Comment and Analysis (Figure 4.11 and Figure 4.12)

From Figure 4.11 we can see that YIN performs badly under white noise, which has the least influence on enhanced Martin's algorithm and SWIPE. After comparing the spectrograms of the five noise types, we conclude that **YIN is highly sensitive to high-frequency components in the noise**. **This makes sense again because YIN is correlation-like. High-frequency noise totally smoothes the audio signals.**

YIN performs best under `babble.wav` noise which doesn't have many high-frequency components. In contrast, the other four do, and most notably in `factory2.wav` (sounds from cut wheel machines). See Figure 4.10.
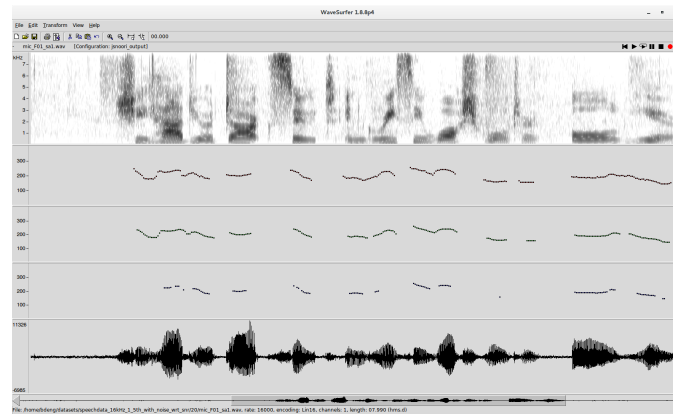
(a) SNR = 20 dB
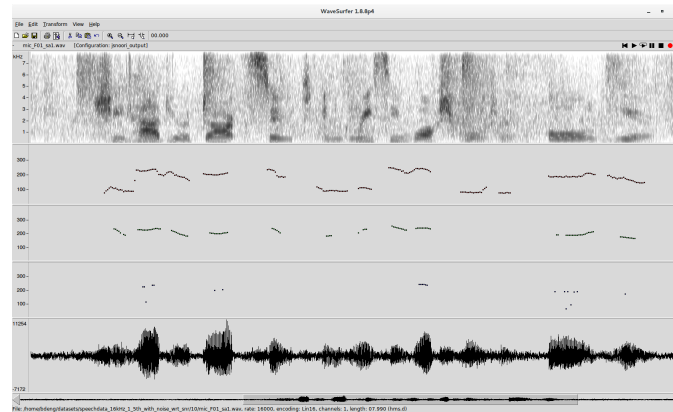
(b) SNR = 10 dB
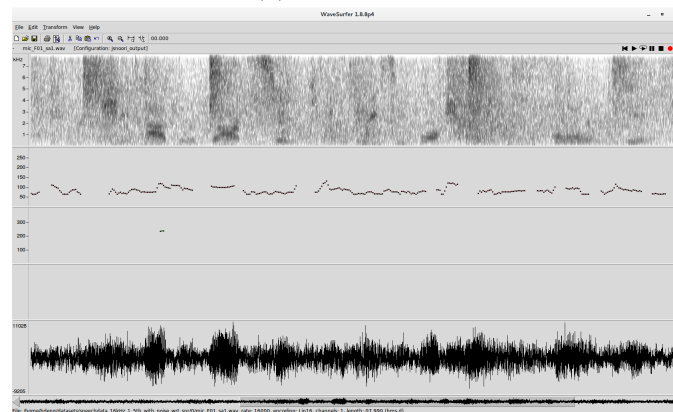
(c) SNR = 0 dB

Figure 4.8: Spectrogram, three plots of estimated F0 values (by Martin's, SWIPE and YIN in order), and waveform shown in Wavesurfer. From top to bottom, signal-to-noise ratios are 20 dB, 10 dB and 0 dB, respectively.
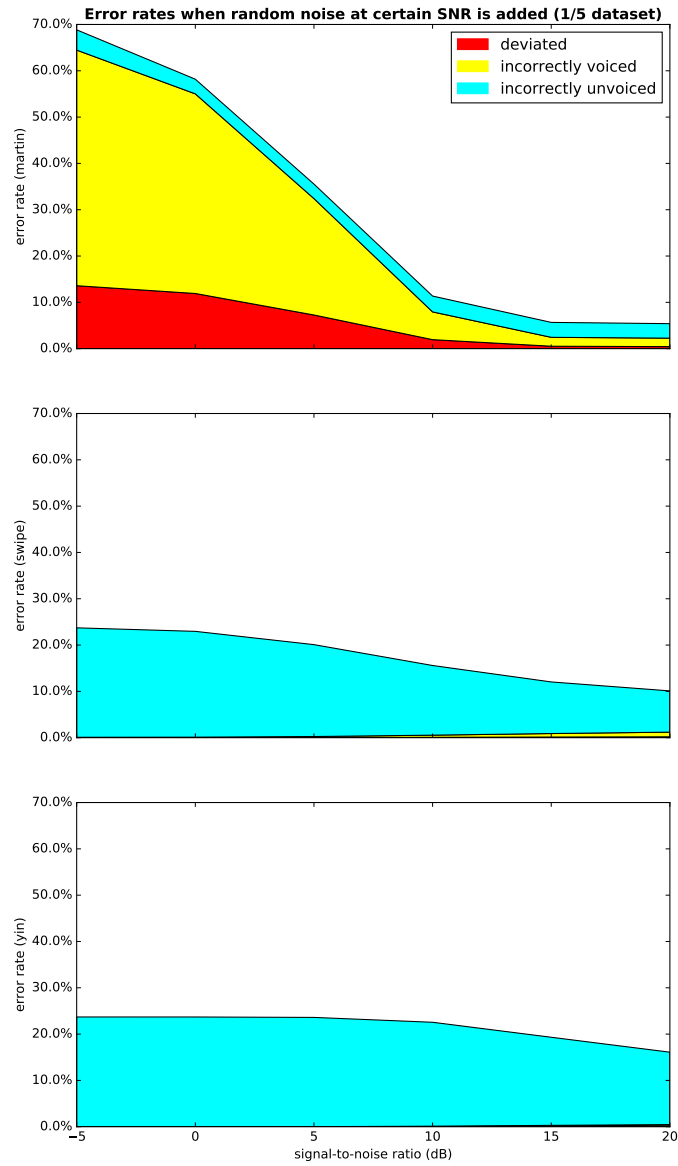
Figure 4.9: Stack plot of the three error types (red for "deviated", yellow for "incorrectly voiced" and cyan for "incorrectly unvoiced").

31
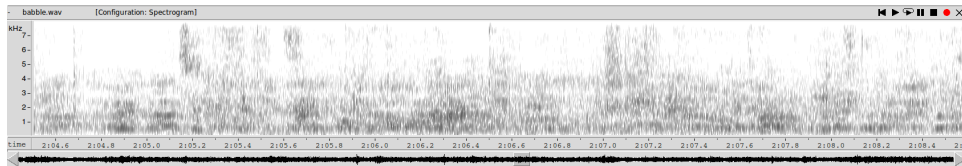
(a) `babble.wav`
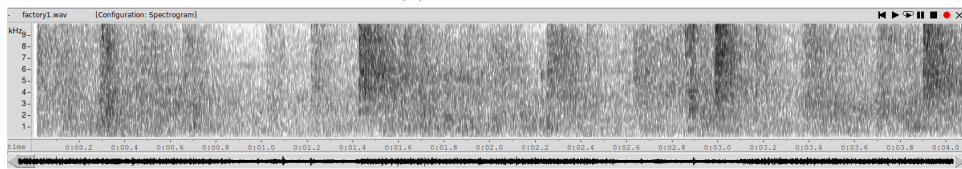
(b) `factory1.wav`
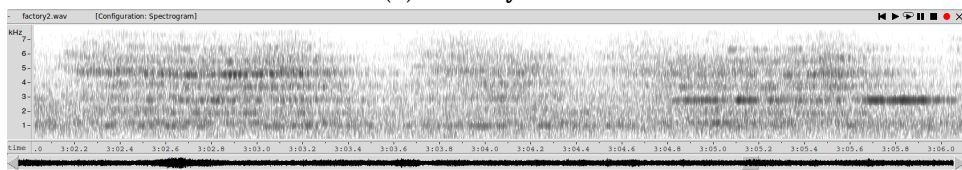
(c) `factory2.wav`
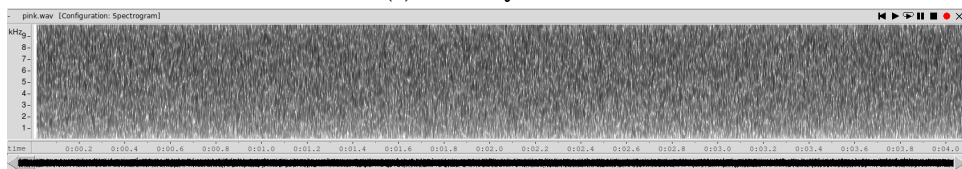
(d) `pink.wav`
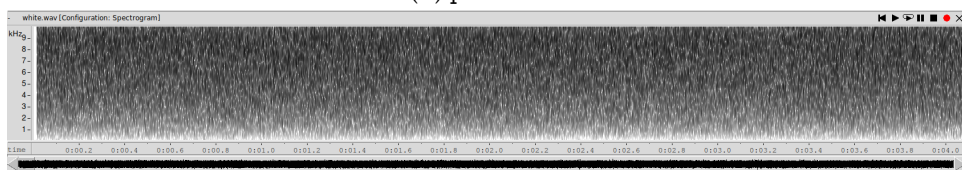
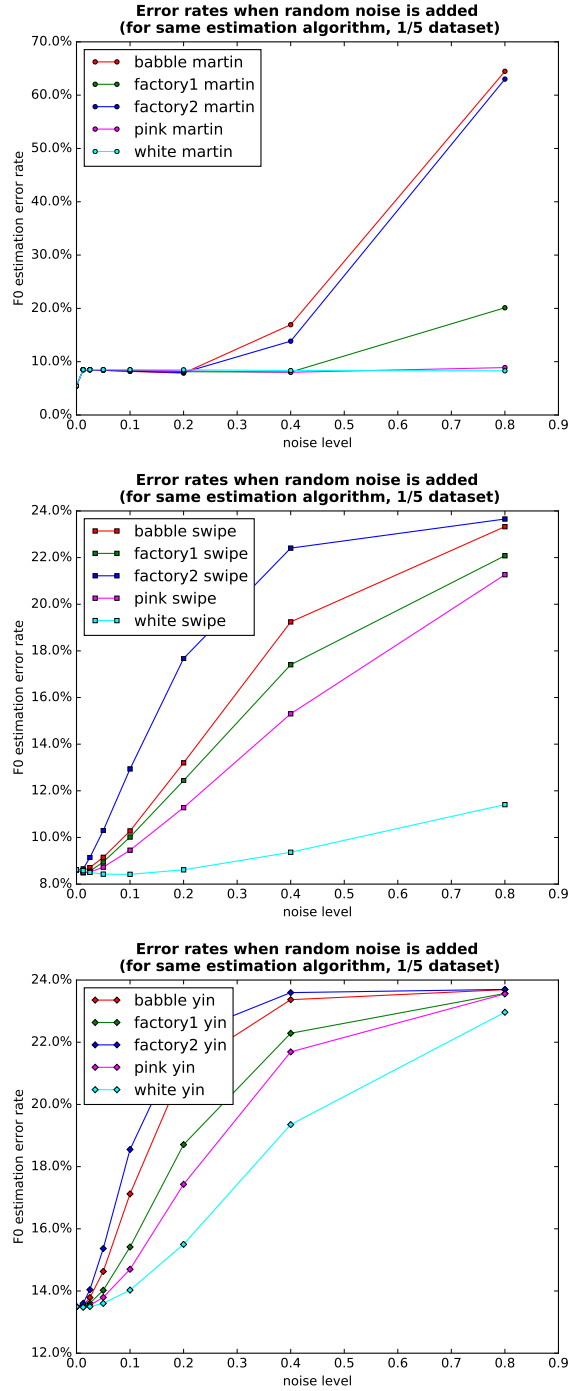(e) `white.wav`

Figure 4.10: Spectrograms of the various types of noise.
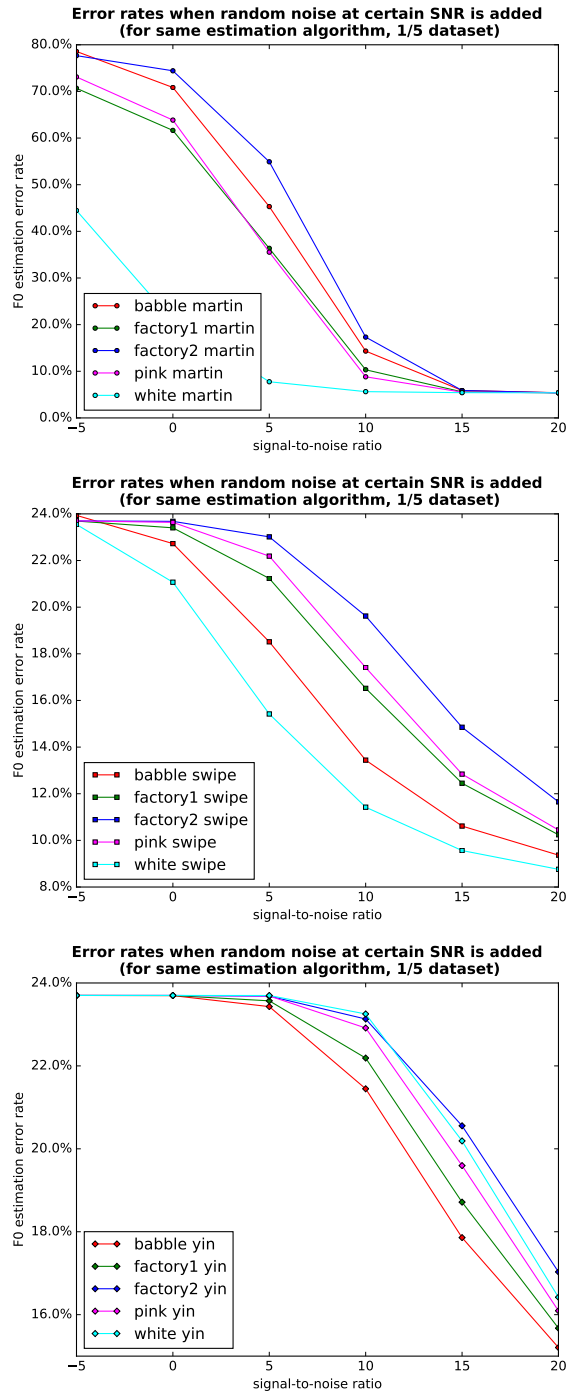
Figure 4.11: F0 estimation error rates for each algorithm (Martin's at the top, SWIPE in the middle, YIN at the bottom) when random noise at certain SNR is sampled from different audio signals. Subfigures have different y-axis limits.
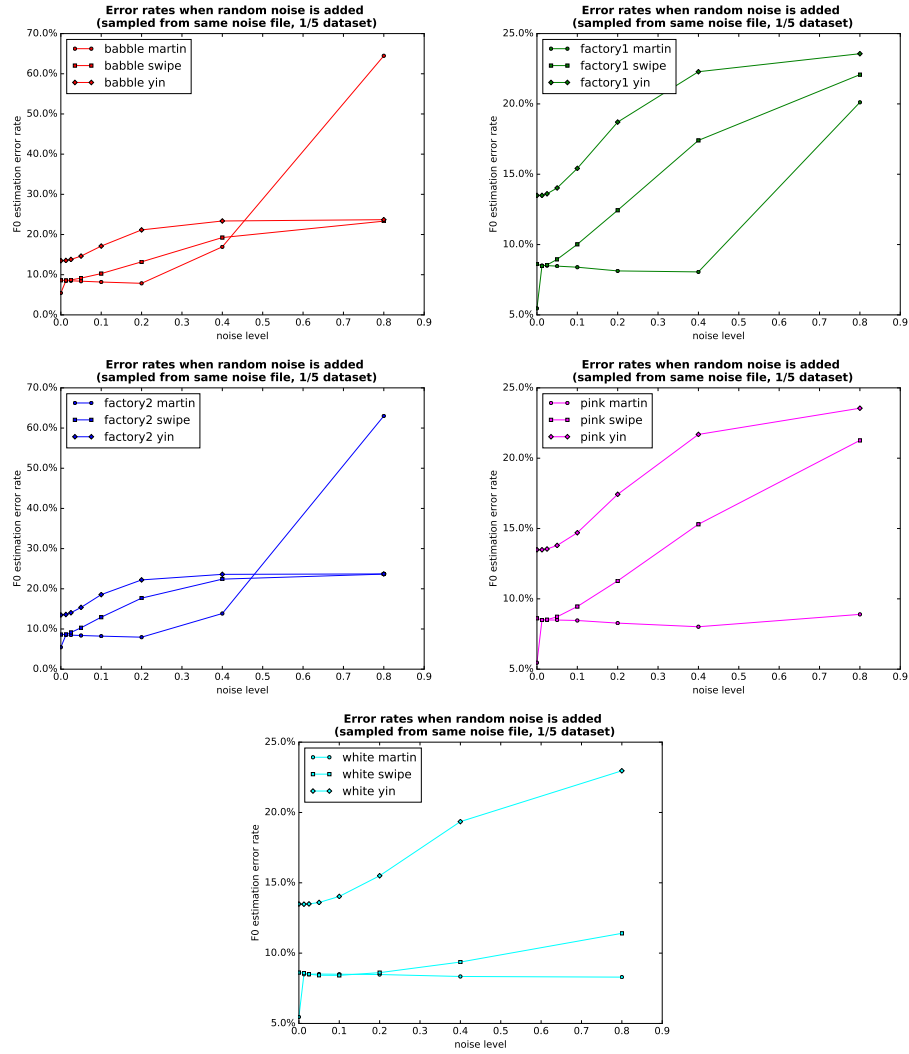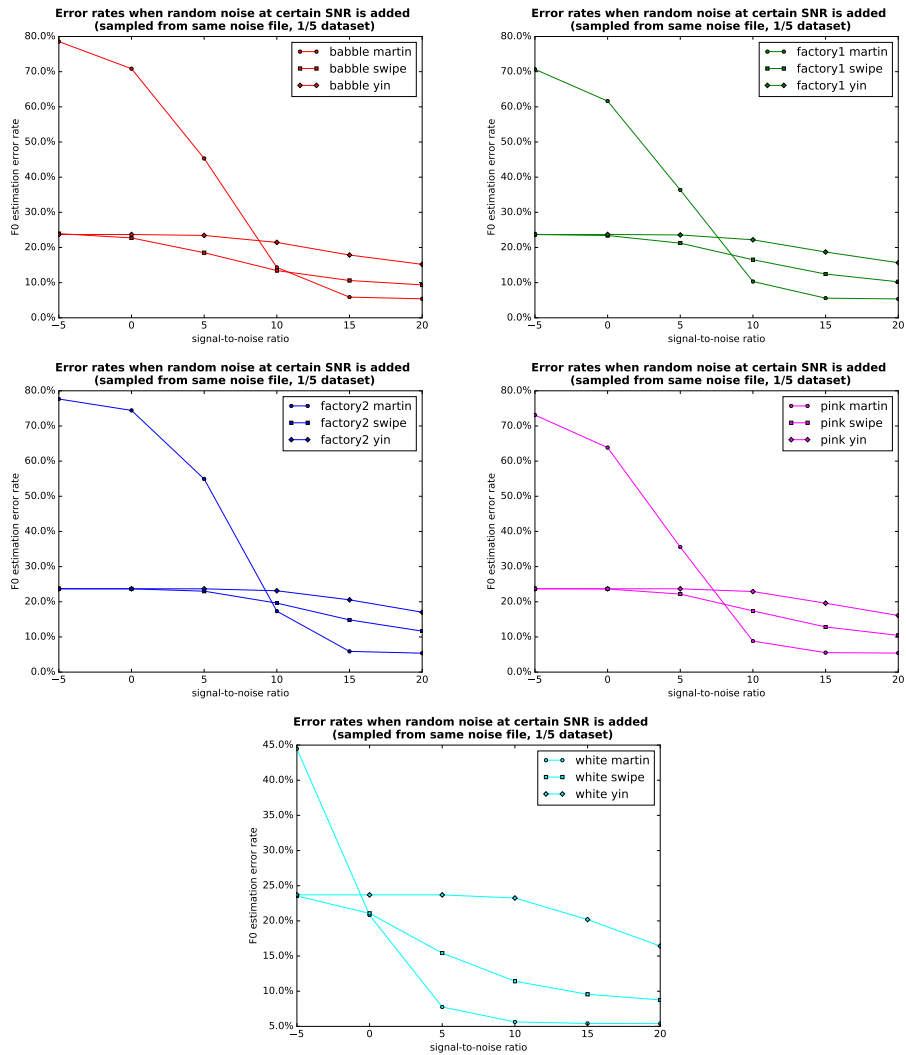
Figure 4.12: F0 estimation error rates of the various algorithms (Martin's represented by circles, SWIPE represented by squares, and YIN by diamonds) for each type of noise at certain SNR. Subfigures have different y-axis limits.

Chapter 5

Confidence Measures on Fundamental Frequency Estimations

As we have stated, none of the existing fundamental frequency estimation algorithms provide a confidence measure.

We treat the prediction of correctness of existing algorithms' estimations as a binary classification problem (correct/incorrect) and design neural network classifiers to output a confidence value, based on the theorem introduced above. Of course, additional information besides estimated fundamental frequency is needed to make the classifiers accurate enough.

5.1   Features and labels

Each run of the fundamental frequency estimation algorithms generates a sample (feature-label pair) for supervised learning and for evaluation.

Features:

Various features are used by the classifiers, some of which are intermediate results computed during estimations, in total there are 59 dimensions:

- energy of the signal

- three ranked candidate results, along with their correlation values

- 40 cepstral coefficients and 12 Mel-frequency cepstral coefficients (MFCCs) (Mermelstein 1976; Davis and Mermelstein 1980)

Labels:

Labels are obtained when calculating error rates during the empirical analyses. We use 0 for incorrect and 1 for correct.

5.2 Neural network architectures

The underlying neural network architectures we designed are multilayer perceptron (MLP) and recurrent models with long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) to incorporate information from context while avoiding the problem with long-term dependencies.

5.2.1 MLP classifier

MLP classifier (Figure 5.1):

- two dense layers of hidden units with rectified linear unit (ReLU) activation function

- a single output unit with `sigmoid` activation

- dropout layers to prevent overfitting
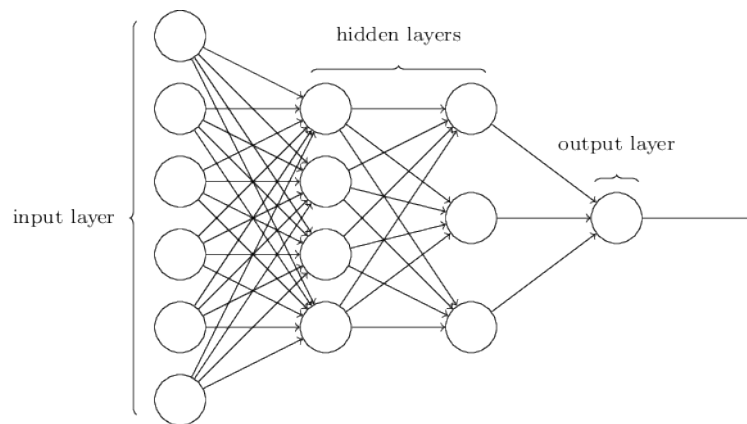
Figure 5.1: Diagram of the MLP classifier.

Figure 5.2 is the computation graph of our MLP classifier, given by the neural network library we use. Data flow from top to bottom.

Tuples like `(None, 59)` provide information about each layer:

- The first element is the batch size. `None` indicates that any positive integer may be expected. But people usually use a power of 2 as batch size so that it is computationally efficient.

- The second element is the dimension of input/output variable (for output variable it's also the number of nodes in that layer).

dense_input_1 (InputLayer)	input:	(None, 59)
	output:	(None, 59)

dense_1 (Dense)	input:	(None, 59)
	output:	(None, 128)

dropout_1 (Dropout)	input:	(None, 128)
	output:	(None, 128)

dense_2 (Dense)	input:	(None, 128)
	output:	(None, 128)

dropout_2 (Dropout)	input:	(None, 128)
	output:	(None, 128)

dense_3 (Dense)	input:	(None, 128)
	output:	(None, 1)
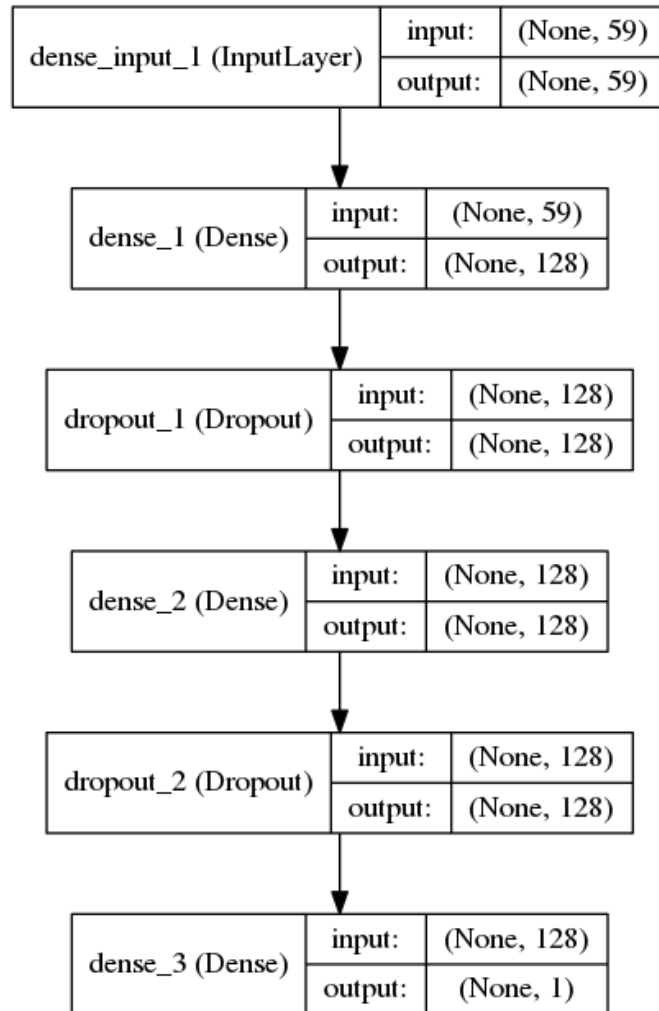
Figure 5.2: Computation graph of the MLP classifier.

5.2.2 LSTM classifier

LSTM classifier:

- two LSTM recurrent layers which encode temporal information and feed sequences of activation values into the next layers

- a dense layer assigns weights to values in the whole sequence

- a single output unit with `sigmoid` activation

- dropout layers to prevent overfitting

Its computation graph is shown in Figure 5.3.

Now the tuples have three elements:

- The first element is still the batch size.

- The second element is the input sequence length (number of time steps). For an input sequence of length n, we think the last sample has temporal dependencies upon samples of the previous $n-1$ time steps. This is a hyperparameter and we try different values of it.

- Input/output variable dimension moves to the third position.

A "Flatten" layer concatenates vectors in the sequence into a single one, for the subsequent non-recurrent layer.

5.2.3   Other architectures

We also investigated other classic architectures for sequence classification, for example, using a max-pooling or average-pooling layer to aggregate the sequence outputs of the LSTM cells, which is popular in the sentiment analysis task.

But in our experiments, despite prolonged tuning, it never outperforms the MLP model.

The cause might be that neighboring time steps (neighboring estimations in our case) have much more complex patterns of dependency, while equal treatment and simple aggregation won't be able to discover them and actually hides the information from upcoming layers.
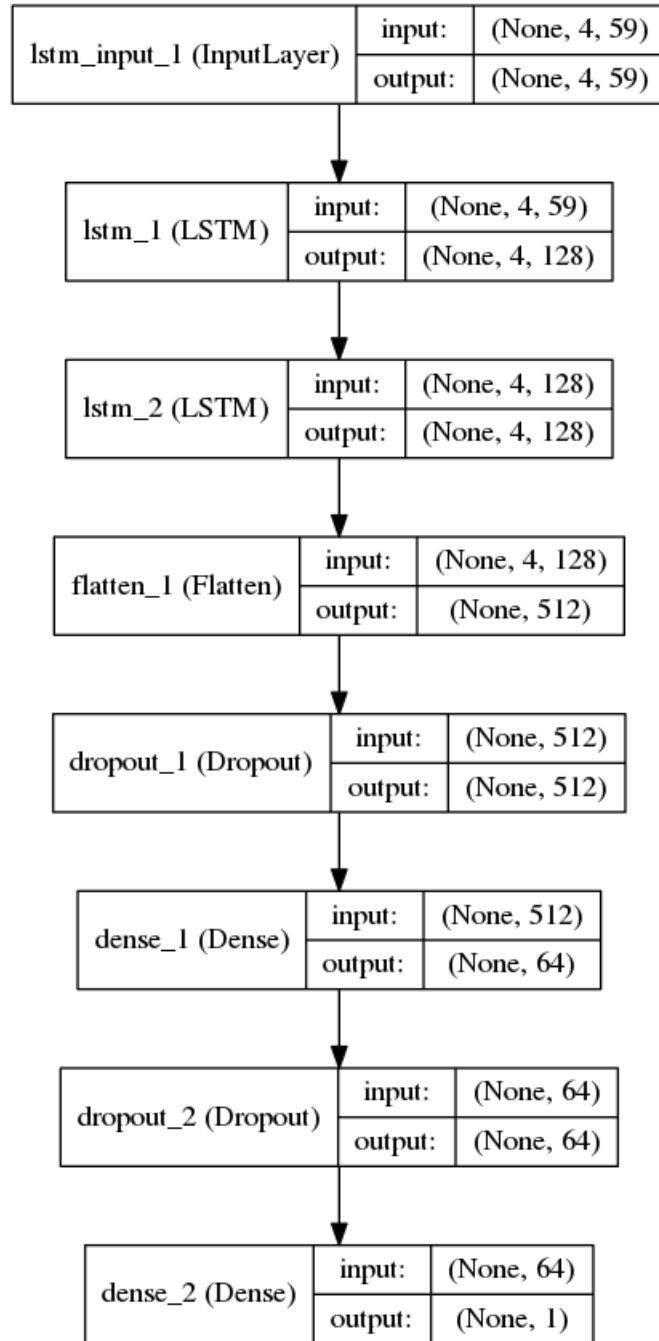
Figure 5.3: Computation graph of the LSTM classifier.
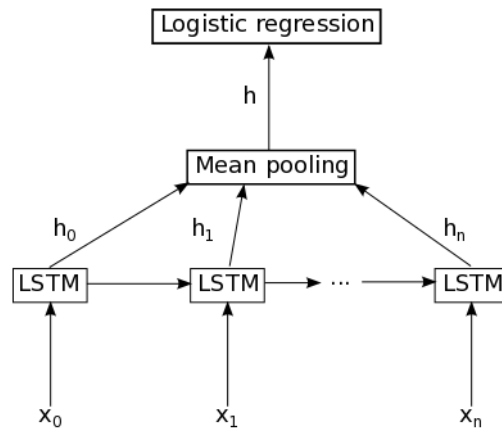
Figure 5.4: Mean-pooling (average-pooling) before output layer.

Chapter 6

Experiments

6.1 Data preparation

The fundamental frequency estimation algorithms have relatively low error rates on clean signals. If we train only on the original dataset, the resulting classifiers would be biased because of lack of negative samples.

So during training and testing, distorted audio files obtained during the empirical analyses (signal-to-noise ratio at $-5\,$dB) are used in addition to the original dataset to balance the number of positive and negative samples. In total, there are approximately 7 million samples for each fundamental frequency estimation algorithm.

After preprocessing, they are stored in matrix-like data structures where each row is a sample. An auxiliary index indicates which audio file they come from. Within the same audio file, samples are sorted in chronological order. This will be useful if we want to explore temporal dependencies.

Then the dataset is split into training and testing sets (80 %/20 %).

- MLP classifier: random sampling on all rows, so the samples are shuffled.

- LSTM classifier: in order to maintain the temporal order of samples and avoid constructing invalid sequences across the boundaries of audio files, the split is done on the audio file level. Rows from the first 80 % audio files are taken out as a whole as the training set.

  Then, within each audio file, iterate through the rows and construct "sequential data" by prepending samples of the past time steps. Length of the sequences is a hyperparameter, as we have stated.

Furthermore, the last 20 % of the training set are taken out and used as the validation set.

```
                         7296   -2.780     ...       0.329 -0.292  0.722
                         7306   -2.518     ...       0.338  0.112  0.457
                         7316   -2.705     ...       0.363 -0.255  0.493
                         7326   -3.179     ...       0.344 -0.336  0.356
                         7336   -3.428     ...       0.441 -0.265  0.107

                                mfcc7  mfcc8  mfcc9  mfcc10  mfcc11  mfcc12  \
                         #time_ms
mic_F01_sa1              16      0.501  0.721 -0.440  -0.166   0.846  -0.460
                         26      1.409  0.103  0.223  -0.242   0.523  -1.144
                         36      0.616 -0.096 -1.322   0.560  -0.281  -0.127
                         46     -0.594 -1.807 -1.426  -0.537  -1.731  -0.492
                         56      0.479 -0.594 -1.118  -1.205  -1.067  -0.790
                         66      0.205  1.034  0.703   1.265  -0.376  -0.487
                         76     -0.347 -0.209  0.086  -0.977  -1.785  -1.392
                         86      0.275  0.322 -0.286  -0.140  -0.449  -0.678
                         96      1.030 -0.850 -0.850  -0.736   0.162  -0.938
                         106     0.243  0.287  0.079   0.135   0.022  -0.477
...                             ...    ...    ...     ...     ...     ...
mic_M10_sx452_distorted 7246    0.125  0.421  0.018   0.435   0.061   0.327
                         7256   -0.004  0.250  0.049  -0.017  -0.286  -0.028
                         7266    0.106  0.489  0.046   0.389  -0.278  -0.428
                         7276    0.047  0.491 -0.165  -0.158   0.333   0.368
                         7286    0.094 -0.146 -0.446   0.172  -0.078  -0.091
                         7296    0.580  0.765 -0.189   0.284  -0.376  -0.690
                         7306    0.324  0.632 -0.058  -0.370  -0.257  -0.253
                         7316    1.001  0.385 -0.251   0.149  -0.192  -0.418
                         7326   -0.041  0.044  0.102   0.459   0.133   0.018
                         7336   -0.041 -0.562 -0.227   0.459  -0.129  -0.387

                                correctness
                         #time_ms
mic_F01_sa1              16             True
                         26             True
                         36            False
                         46            False
                         56            False
                         66            False
                         76            False
                         86            False
                         96             True
                         106            True
...                                     ...
mic_M10_sx452_distorted 7246          False
                         7256          False
                         7266          False
                         7276          False
                         7286          False
                         7296          False
                         7306          False
                         7316          False
                         7326          False
                         7336          False

[6858022 rows x 60 columns]
>>>
```

Figure 6.1: A very small part of the `pandas` DataFrame storing our data for Martin's algorithm. There are in total 60 columns (59-dimensional features plus one-dimensional label). The two-level hierarchical indexing consists of audio file basename and time offsets within the audio file. Audio files with additive noises have a "`_distorted`" suffix.

6.2   Training

The models are trained using stochastic gradient descent (SGD) on mini-batches with binary cross-entropy as objective function. Class probability threshold for classification is set to 0.5 consistently.

Various techniques from (LeCun et al. 2012) and (Orr and Müller 2003) are applied, though nowadays they are becoming standard procedures.

- normalize the training data to zero mean, unit variance on each dimension

- learning rate decay

- Nesterov momentum (Nesterov 1983)

- L1 or L2 regularization on the weights

- early-stopping using the validation set (if the performance on validation set drops after an epoch, stop updating the parameters immediately)

The training takes several hours, up to half a day for each model and hyperparameter setting.

6.3   Testing

When normalizing the training data to zero mean, unit variance on each dimension, we remove the mean value of each feature, then scale by dividing non-constant features by their standard deviation.

The resulting scaling, determined by such mean values and standard deviations calculated on each dimension, must be applied to testing data or new data in production use before feeding them into the models.

In this way, the very important assumption in machine learning that training and testing data are sampled from the same distribution still holds, because the same transformation is applied.

After transformation, the testing data is fed into the models. The outputs will be used for evaluation of the models.

Chapter 7

Results and Evaluation

We evaluate the performance of the classifiers trained on every algorithm's data by

- test accuracy

- area under the receiver operating characteristic curve (ROC AUC)

- normalized mutual information (NMI)

> **What needs to be clarified is that performance of the classifiers reflects qualities of the generated confidence measures, they are not necessarily related to performance of the fundamental frequency estimation algorithms.**

7.1   Metrics

There are four possible outcomes from a binary classifier, identified by comparing predicted results with ground truth.

- true positive, predicted as positive and actually positive

  (The classifier predicts that the F0 estimation is correct, and the F0 estimation is actually correct.)

- false positive, predicted as positive but actually negative

  (The classifier predicts that the F0 estimation is correct, but the F0 estimation is actually incorrect.)

- false negative, predicted as negative but actually positive

  (The classifier predicts that the F0 estimation is incorrect, but the F0 estimation is actually correct.)

- true negative, predicted as negative and actually negative

  (The classifier predicts that the F0 estimation is incorrect, and the F0 estimation is actually incorrect.)

We use tp, fp, fn, tn to denote the number of samples in corresponding categories respectively. Other metrics are constructed based on these values.

7.1.1 Test accuracy

Test accuracy is the overall accuracy over the testing set.

$$accuracy = \frac{tp + tn}{tp + fp + fn + tn}$$

The ideal value is 1.

In our binary classification problem, "positive" means "fundamental frequency estimation result is correct" and "negative" means "fundamental frequency estimation result is incorrect".

During the calculation of this metric, class probability thresholds are set to 0.5 for both classes, which means if the `sigmoid` activation value is greater than 0.5 then sample is classified as "positive", otherwise "negative".
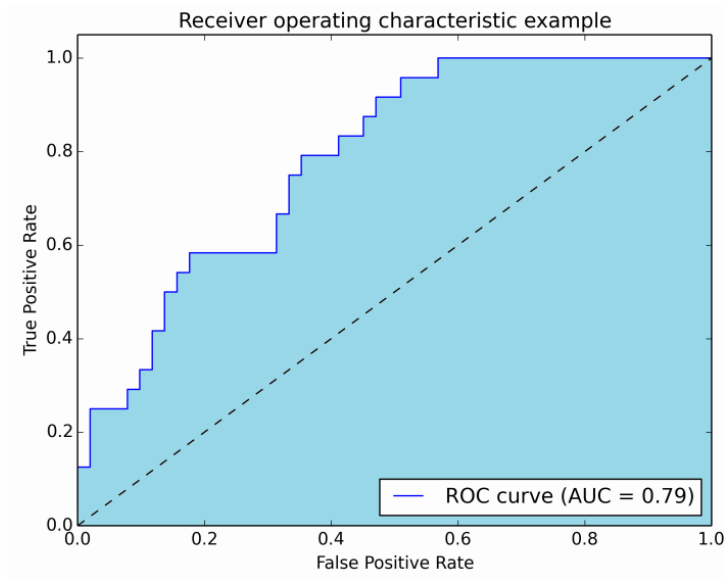
7.1.2 Receiver operating characteristic curve

Figure 7.1: Example ROC curve and its AUC.

The receiver operating characteristic (ROC) curve plots true positive rates versus false positive rates for a classifier capable of outputting class probabilities results.

$$true\ positive\ rate(tpr) = \frac{tp}{tp + fn}$$
$$false\ positive\ rate(fpr) = \frac{fp}{fp + tn}$$

Points on the ROC curve are obtained by setting different thresholds on class probabilities when making classification decisions (instead of using a fixed threshold, which is set to 0.5 in the previous section). For example, assuming that a classifier outputs a confidence measure of 0.7 for a given sample, with a threshold of 0.6 the output will be assumed "positive", whereas with a threshold of 0.8, the output will be assumed "negative".

Area under the ROC curve can be interpreted as the probability that the classifier will assign a higher score to a randomly chosen positive sample than to a randomly chosen negative sample. The ideal value is also 1. In reality, only a finite number of points on the curve can be obtained, so we construct trapezoids under the curve as an approximation to area.

As our confidence measures are equal to the positive class probabilities, area under the ROC curve reflects objectively their performance over the whole value range (0 to 1).

7.1.3 Normalized mutual information

Normalized mutual information (NMI) measures agreement of class assignments.

The entropy of a discrete random variable X is defined as

$$H(X) = -\sum_{x \in X} P(x) \log P(x).$$

The mutual information of two discrete random variables X and Y, measuring mutual dependence, is defined as

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} P(x,y) \log \left(\frac{P(x,y)}{P(x)P(y)} \right).$$

NMI is obtained by dividing mutual information by the geometric mean of two random variables' entropies. Its range is between 0 and 1 and again 1 is ideal.

$$\text{NMI} = \frac{I(X;Y)}{\sqrt{H(X)H(Y)}}$$

Both the classification results and ground truth can be seen as discrete random variables with possible values $\{0, 1\}$ and a probability mass function P. Then NMI can be applied. This metric is furthermore symmetric. So it doesn't matter which one of the two X and Y represent.

7.2 Input sequence length for LSTM model

During our experimentation, a longer input sequence length (more time steps) in LSTM model gives better results, as shown in Table 7.1.

Model	Tuned for	Accuracy	ROC-AUC	NMI
	Martin's	0.926	0.968	0.641
LSTM (sequence length: 2)	SWIPE	0.880	0.870	0.219
	YIN	0.893	0.874	0.230
	Martin's	0.932	0.971	0.659
LSTM (sequence length: 3)	SWIPE	0.883	0.878	0.238
	YIN	0.901	0.889	0.272
	Martin's	0.934	0.971	0.666
LSTM (sequence length: 4)	SWIPE	0.886	0.880	0.247
	YIN	0.902	0.892	0.276

Table 7.1: Performance of the LSTM model when different input sequence lengths are applied during training and testing.

Each line in the table gives the model's classification performance when trained on one of the fundamental frequency estimation algorithms' data.

The increasing performance proves that there are indeed long-term dependencies and our LSTM model is capable of capturing them. **It may not be monotonically increasing when the input sequence length grows even longer.**

7.3 MLP model versus LSTM model

Table 7.2 reports best results for both neural network architectures until now. They are achieved when there are 128 nodes in both the MLP model's hidden layers and the LSTM model's recurrent layers. The dense layer in the LSTM model has 64 nodes. Dropout rate is set to 0.5 uniformly.

- Our LSTM model outperforms the MLP model everywhere as expected. This shows the advantage of recurrent neural networks.

Model	Tuned for	Accuracy	ROC-AUC	NMI
MLP	Martin's	0.916	0.957	0.585
	SWIPE	0.874	0.867	0.209
	YIN	0.863	0.829	0.143
LSTM (sequence length: 4)	Martin's	0.934	0.971	0.666
	SWIPE	0.886	0.880	0.247
	YIN	0.902	0.892	0.276

Table 7.2: Metrics of the best-performing classifiers.

- High accuracy and ROC-AUC values guarantee that our models will be highly applicable in production use.

- There are no baselines to compare with. As mentioned at the beginning of the thesis, there were no confidence measures proposed on estimated F0 values in the literature.

- NMI results indicate that there is still margin for improvement regarding class assignment. A certain type of fundamental frequency estimation error is "dominating" the negative samples, making it harder for classifiers to generalize to tell which are wrong estimations. It may just be "incorrectly voiced" error for enhanced Martin's algorithm and "incorrectly unvoiced" error for SWIPE and YIN.

- For now, our models are universal for any fundamental frequency estimation algorithm. If we combine multiple algorithms, special settings for certain algorithms, for example, a decision tree based on signal energy may help detect SWIPE and YIN's potential "incorrectly unvoiced" errors, then the system can relies more on Martin's estimation. But at the same time we need to make sure the overall classifiers still estimate Bayesian probabilities.

7.4   Performance on distorted audio signals

To further investigate the robustness of our models, the distorted audio corpus at different signal-to-noise ratios are used as extra testing data.

For all combinations of fundamental frequency estimation algorithms and signal-to-noise ratios, the metrics of our MLP model and LSTM model are plotted together and compared. Classification accuracy, ROC AUC and NMI results are shown in Figure 7.2, Figure 7.3 and Figure 7.4 respectively.

- Our models generalize well on new data. There is no sharp decrease in performance when the signal-to-noise ratio changes.

- LSTM model outperforms MLP model most of the time, which confirms again the existence of temporal dependencies in our problem.

- Again, the NMI results have potential for improvement.

7.5 Performance on voiced and unvoiced segments

Samples can be categorized into voiced and unvoiced segments, based on whether the reference fundamental frequency is greater than zero or not.

The three metrics are then calculated on voiced and unvoiced segments separately. Now four curves are plotted at a time. Results are shown in Figure 7.5, Figure 7.6 and Figure 7.7.

- Generally speaking, our classifiers work better on unvoiced segments. One possible explanation is that the size of feature space (contains all possible values of the 59-dimensional machine learning features) is much smaller for unvoiced segments than for voiced segments, so it's easier for a classifier to generalize on data for unvoiced segments.

- Non-monotonicity in the curves indicates fluctuation in the size of feature space. The classification problem is most difficult when there is a moderate amount of noise.

- Training on data at certain signal-to-noise ratio might be necessary depending on use cases, whether the user cares more about voiced or unvoiced segments. Because intersecting curves are observed.
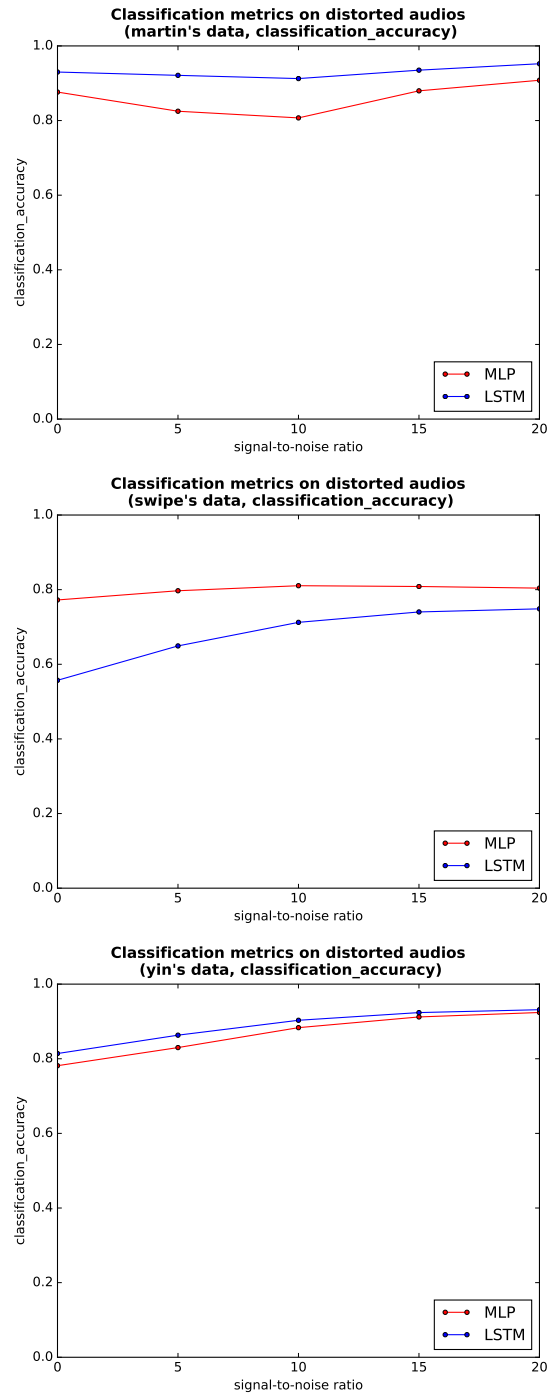
Figure 7.2: MLP model and LSTM model's accuracies on distorted testing audio signals.
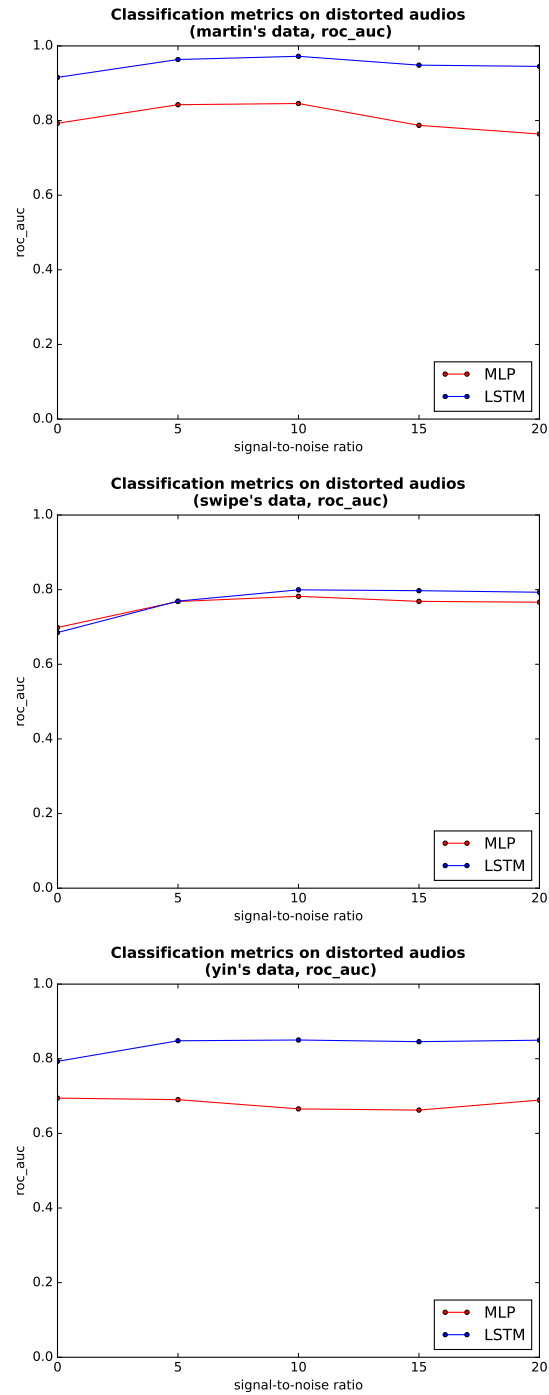
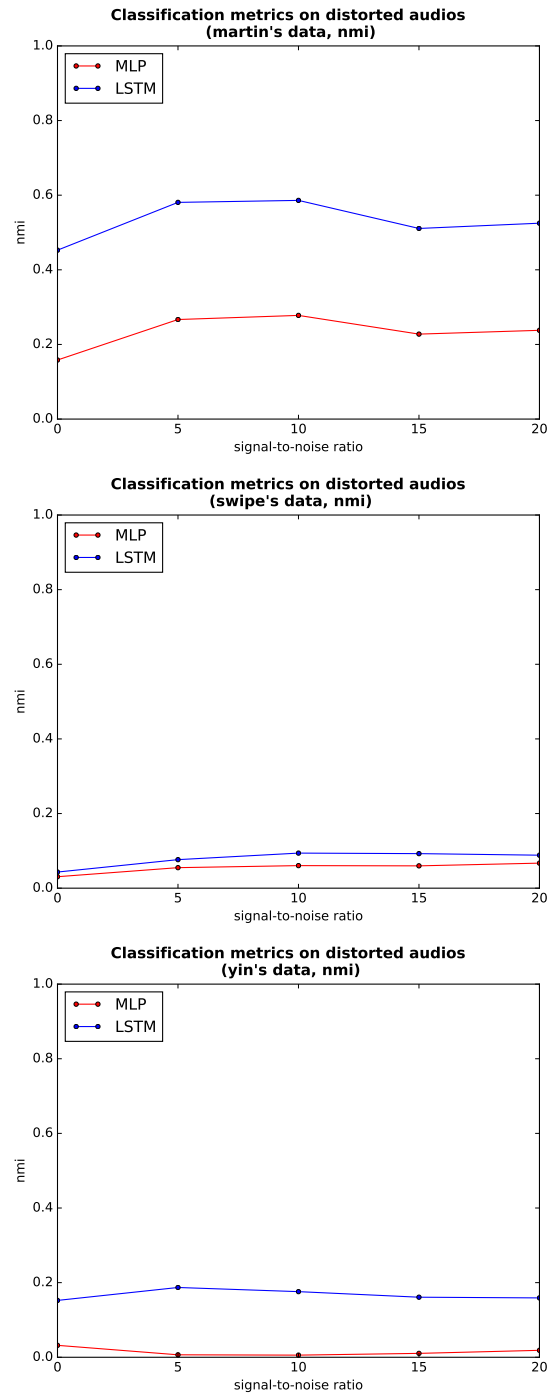Figure 7.3: MLP model and LSTM model's ROC AUCs on distorted testing audio signals.

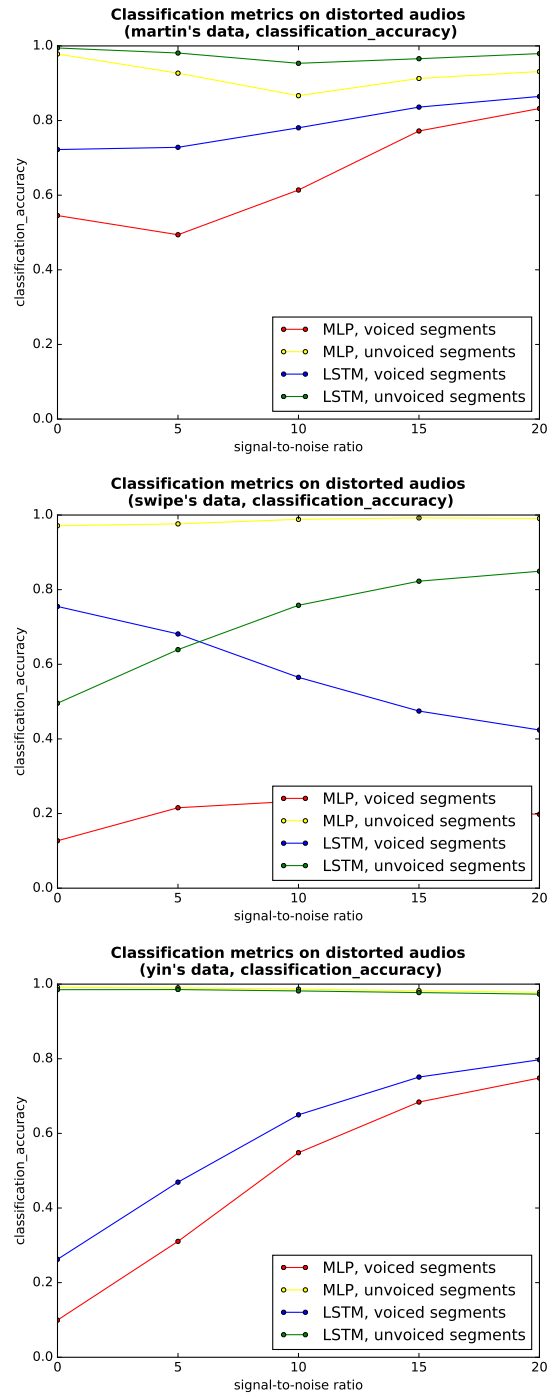Figure 7.4: MLP model and LSTM model's NMIs on distorted testing audio signals.

Figure 7.5: MLP model and LSTM model's accuracies on distorted testing audio signals (voiced and unvoiced segments).
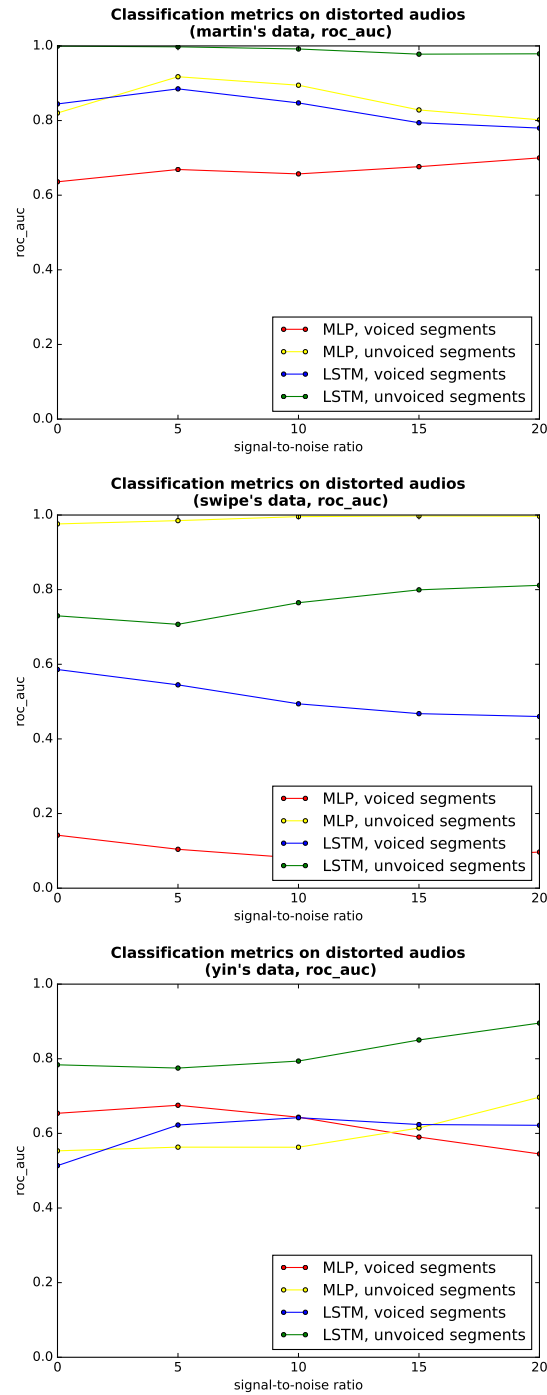
Figure 7.6: MLP model and LSTM model's ROC AUCs on distorted testing audio signals (voiced and unvoiced segments).
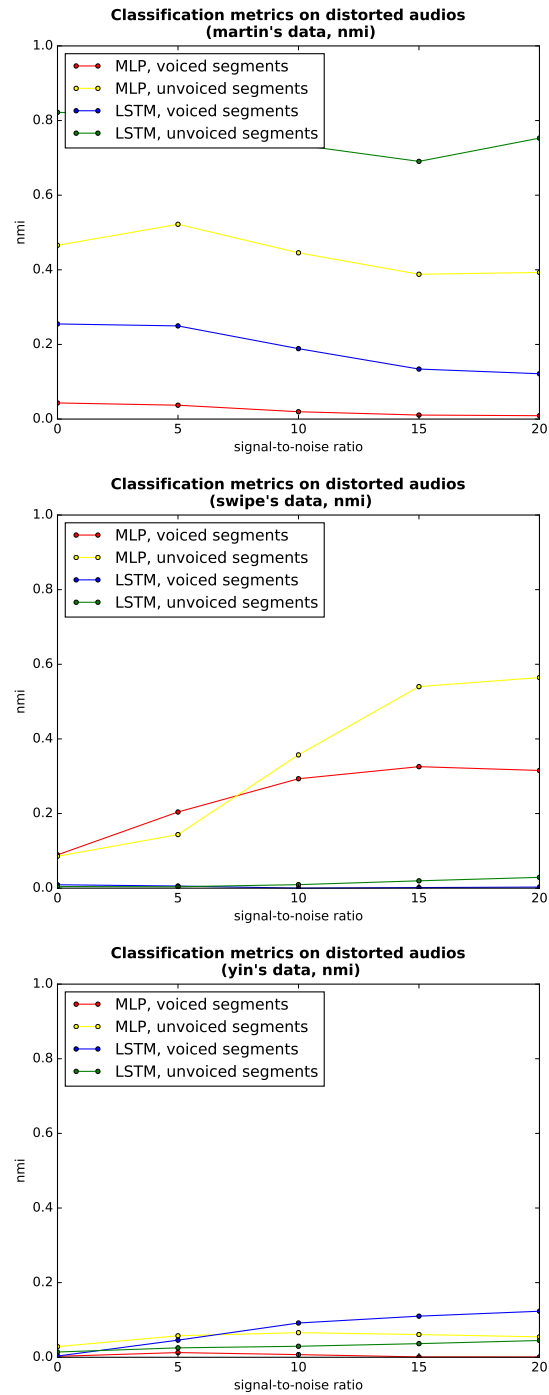
Figure 7.7: MLP model and LSTM model's NMIs on distorted testing audio signals (voiced and unvoiced segments).

Chapter 8

Conclusion

We are the first in this field to tackle the problem of designing confidence measures for fundamental frequency estimations.

Our outcome is a universally applicable yet customizable and extensible neural network-based framework able to output confidence values directly. There is no need to modify the target fundamental frequency estimation algorithm, as long as the estimated frequencies and other features are involved in training.

Its excellent performance is demonstrated on several existing fundamental frequency estimation algorithms, which shows its utility value.

Future work

- Bidirectional LSTMs may work even better by also looking "forwards" in the sequences. It's reasonable to assume that there are temporal dependencies on future time steps.

- Neural network regression models are able to output an estimated frequency along with the confidence, like an ensemble method, and estimated frequencies from various algorithms can be used as features.

- We may further investigate the possibility of using a convolutional neural network (CNN) to calculate the fundamental frequency values, working directly on the spectrogram.

Appendix A

Appendix

We use `pandas` (McKinney et al. 2010) to do data preparation and `matplotlib` (Hunter et al. 2007) for plotting. Neural networks are implemented using `Python` and `Keras` (Chollet 2015) library.

All the code used in this work is hosted at `https://github.com/bryandeng/f0-cm`, along with necessary documentation.

Some of the experiments are run on Grid'5000 (Balouek et al. 2012).

Bibliography

Balouek, Daniel et al. (2012). "Adding virtualization capabilities to the Grid'5000 testbed". In: *Cloud Computing and Services Science*. Springer, pp. 3–20.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult". In: *Neural Networks, IEEE Transactions on* 5.2, pp. 157–166.

Camacho, Arturo (2007). "SWIPE: A sawtooth waveform inspired pitch estimator for speech and music". PhD thesis. University of Florida.

Chollet, François (2015). *Keras*. `https://github.com/fchollet/keras`.

Davis, Steven B and Paul Mermelstein (1980). "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences". In: *Acoustics, Speech and Signal Processing, IEEE Transactions on* 28.4, pp. 357–366.

De Cheveigné, Alain and Hideki Kawahara (2002). "YIN, a fundamental frequency estimator for speech and music". In: *The Journal of the Acoustical Society of America* 111.4, pp. 1917–1930.

Garofolo, John S et al. (1993). "DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1". In: *NASA STI/Recon Technical Report N* 93.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hunter, John D et al. (2007). "Matplotlib: A 2D graphics environment". In: *Computing in science and engineering* 9.3, pp. 90–95.

Jiang, Hui (2005). "Confidence measures for speech recognition: A survey". In: *Speech communication* 45.4, pp. 455–470.

LeCun, Yann A et al. (2012). "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, pp. 9–48.

Martin, Philippe (1982). "Comparison of pitch detection by cepstrum and spectral comb analysis". In: *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'82.* Vol. 7. IEEE, pp. 180–183.

McKinney, Wes et al. (2010). "Data structures for statistical computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Vol. 445, pp. 51–56.

Mermelstein, Paul (1976). "Distance measures for speech recognition, psychological and instrumental". In: *Pattern recognition and artificial intelligence* 116, pp. 374–388.

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814.

Nesterov, Yurii (1983). "A method of solving a convex programming problem with convergence rate O (1/k2)". In: *Soviet Mathematics Doklady*. Vol. 27. 2, pp. 372–376.

Orr, Genevieve B and Klaus-Robert Müller (2003). *Neural networks: tricks of the trade*. Springer.

Pirker, Gregor et al. (2011). "A Pitch Tracking Corpus with Evaluation on Multipitch Tracking Scenario." In: *INTERSPEECH*, pp. 1509–1512.

Richard, Michael D and Richard P Lippmann (1991). "Neural network classifiers estimate Bayesian a posteriori probabilities". In: *Neural computation* 3.4, pp. 461–483.

Sjölander, Kåre and Jonas Beskow (2000). "Wavesurfer-an open source speech tool." In: *Interspeech*, pp. 464–467.

Srivastava, Nitish et al. (2014). "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Talkin, David (1995). "A robust algorithm for pitch tracking (RAPT)". In: *Speech coding and synthesis* 495, p. 518.

Varga, Andrew and Herman JM Steeneken (1993). "Assessment for automatic speech recognition: II. NOISEX-92: A database and an experiment to study the effect of additive noise on speech recognition systems". In: *Speech communication* 12.3, pp. 247–251.