Melodic Phrase Segmentation Using Conditional Random Field

Devin Kwok Yifan Xie Wisang Sugiarta Yihong Wu

Abstract

Melodic phrase segmentation is the task of identifying which notes in a melody are the boundaries of musical phrases. This task has applications in automatic music detection and composition as well as for developing musical databases. Previous works has found that the linear-chain conditional random field (CRF) combined with CNN or LSTM features has high performance for melodic phrase segmentation. We reimplement a linear-chain CRF following (Sutton & McCallum, 2006) and use it to investigate the effect of different feature vectors on CRF classification performance. In particular, we consider whether hand-crafted features are competitive with LSTM-dervied features using various phrase labelling approaches from (Zhang & Xia, 2020). We find that hand-crafted features and labelling schemes do not perform better than a naive categorical encoding of pitch and duration for adjacent notes, and that LSTM features significantly outperform this baseline.

1. Introduction

Melodies and songs have always been of interest to humans. Every culture and population has developed unique combinations of notes and pitches that represent melodic phrases. The task of identifying phrase boundaries is called melodic phrase segmentation. While this task is quite simple for humans to do by ear, representing melodies and detecting changes is a complex computational task (Wiggins, 2016). Automated phrase segmentation is a well established computational problem which can be useful in automated music detection and composition, as well as in creating and managing music databases (Doornbusch, 2010; Hashida et al., 2008). However, representing melodic data effectively is difficult, and feature selection for downstream tasks is usually not trivial. Heuristic-based solutions have been shown to be unstable across different music genres, as notes, note length, and repetition of motifs or n-grams are not consistent features for downstream tasks (Temperley, 2004).

Probabilistic graphical models allow melodies to be represented as series of note-to-note dependencies. Previous work has shown that modelling prior knowledge in sequences and including latent variables in models such as a Hidden Markov Model (HMM) can improve the increase the accuracy of melodic phrase segmentation tasks (Raphael, 1999). Conditional Random Fields (CRFs) are a class of discriminative models which can model the same probability distributions as HMMs, but generally perform better in classification tasks (Sutton & McCallum, 2006). This is because discriminative models maximize the conditional probability of labels given data, rather than the joint probability of labels and data (Sutton & McCallum, 2006). This enables CRFs to perform well even when the correct distribution for modelling dependencies within data is unknown.

In addition to the use of CRFs for melodic phrase segmentation, there has also been recent work in incorporating deep neural networks as input features to CRF models. In particular, Zhang & Xia (2020) use a one-dimensional convolutional neural network (CNN) or bidirectional long short-term memory (LSTM) architecture to transform and encode input melodic phrases into features for a CRF model. They find that these neural network models significantly outperform heuristic-based methods including Grouper (Temperley, 2004) and IOI (Cenkerová et al., 2018), with the bidirectional LSTM having the best overall performance. Additionally, they find that CRFs with neural network features have even better performance than neural networks alone. The authors also conduct experiments to test whether label engineering methods that reducing the sparsity of positive labels can improves performance. In particular, two label engineering methods are introduced: linear ascending labels that restart at phrase boundaries, and exponential labels that peak around phrase boundaries. The authors find that label engineering improves performance over sparse binary labels.

The advantage of incorporating a deep neural network into a CRF model is that a neural network can leverage complex and long-term dependencies in the input sequence. Conversely, incorporating the CRF model over a deep neural network allows strong dependencies between labels to be enforced, leading to better sequence labelling accuracy. For example, a deep neural network may label two adjacent notes as phrase starts with high probability, but the CRF will prevent both notes from being labelled as phrase starts because the transition probability of two adjacent phrase

starts is extremely unlikely.

While Zhang & Xia (2020) clearly show that neural networks combined with CRFs perform well on the phrase segmentation task, they do not examine whether the CRF model can perform well without needing neural network features. We therefore investigate whether a CRF model with hand-crafted features performs more similarly to deep learning methods or heuristic-based methods. In this paper, we first describe the theory of CRFs, and then apply the linear-chain CRF to conduct melodic phrase segmentation on the Essen Folksong Collection (Schaffrath, 1995). We implement and test a linear-chain CRF with features similar to rules used by heuristic methods. Our hand-crafted features are derived from the relative and absolute pitch and duration of adjacent notes, and we compare their performance against bidirectional LSTM features. Additionally, we examine the performance of another label engineering method, which represents musical cadences via truncated linear descending labels towards phrase endings.

2. Melodic Segmentation Task

2.1. Essen Folksong Collection

We use the Essen Folksong Collection (ESFC) [1] as our experimental data. Each melody in ESFC consists of multiple melodic phrases which are labelled at a per-note level. The relative pitch and duration of notes are encoded in a custom text format. We select 5683 melodies from the collection which do not contain irregularities such as zero duration notes or wrongly encoded pitches. To represent each melody, we extract a sequence of note objects with three properties:

Pitch: We map pitch to an integer representing its relative distance in semitones on the chromatic (12 note) scale, where 0 is the reference pitch provided by the dataset for that melody. For example, 12 represents a pitch which is one octave higher than the reference, and -7 represents the interval of a perfect fifth below. Musical rests are set to the "$None$" type in Python.

Duration: We represent duration as a multiple of the shortest rhythmic unit in a given melody. Mathematically, if $m > 0$ is the duration of the shortest note; then the duration of any other note in the melody is $1.5^i 2^j m$ for $i, j \in \mathbb{Z}^+$. For example, if the shortest note in a melody is an eighth note, then it would have a duration of 1, and dotted quarter notes would have a duration of 3.

Phrase label: We represent melodic phrase boundaries as binary labels corresponding to the start of each phrase, where the first note of a phrase is labelled 1 and the remaining notes are labelled 0. The first note of the melody is always labelled 1.

[1] Available for download at http://www.esac-data.org/

2.2. Feature Engineering

To encode notes as features for the linear-chain CRF, we first treat pitch and duration as two categorical variables. This is simply done by converting a canonical (rounded) representation of each value into a string format, and then converting the resulting string into a one-hot vector. The set of possible pitches and durations is extracted from the training data, and any subsequent unknown pitches or durations are encoded as the 0 vector.

We also extract several handcrafted features from the pitch $P(x_t)$ and duration $D(x_t)$ of each note in a melodic sequence. These features are as follows:

1. The interval distance $|P(x_t) - P(x_{t-1})|$ and direction $P(x_t) > P(x_{t-1})$ between adjacent notes (or 0 if either note is a rest).
2. The $\log_2$ ratio of duration between adjacent notes $\log_2 \frac{D(x_t)}{D(x_{t-1})}$. Base 2 is chosen since musical durations typically occur in multiples of 2.
3. Whether a note is a rest, the tonic $\equiv (0 \mod 12)$, or the dominant $\equiv (7 \mod 12)$.
4. Whether the pitch of a note belongs to the tonic, predominant, or dominant function chords (these categories are defined from music theory). Specifically, we consider pitches $\{0, 3, 4, 7\}$ as tonic (C, Eb, E, G in C major), $\{2, 5, 6, 8, 9\}$ as predominant (D, F, F#, Ab, A in C major), and $\{7, 11, 2, 5\}$ as dominant (G, B, D, F in C major).

Features that depend on adjacent timesteps $(t, t-1)$, such as relative pitch and duration ratio, reflect Markov assumptions about the adjacent dependency of notes. We consider several variations of these Markov assumptions which are 1) independent x_t discarding relative pitch and duration, 2) symmetric dependence on adjacent notes in time $x_t|x_{t-1}, x_{t+1}$ and 3) dependence on previous two notes $x_t|x_{t-1}, x_{t-2}$.

To use these features in the linear-chain CRF, we replicate the parameterization of a HMM. First, we choose a subset of the features above for inclusion. Included features are vectorized to form a vector of length F for each position t in the sequence, where K is the total number of numeric features plus the number of categories for every categorical feature. Numeric features are normalized to $[-1, 1]$ and assigned to a component of the feature vector, while categorical features with c categories are one-hot encoded and assigned to c components of the feature vector. We then convert this vector into the analogue of the emission probability

matrix by taking its outer product with an L-dimensional one-hot vector of the label state at position t.

$$f_{\text{emission}} = [f_t(\mathbf{x})\mathbf{1}_{y_t=k}]_{k\in L}$$

We also generate the analogue of the transition probability matrix by taking the outer product of the one-hot encodings of the label state at position t and $t-1$.

$$f_{\text{transition}} = \left[\mathbf{1}_{y_t=j}\mathbf{1}_{y_{t-1}=k}\right]_{\{j,k\}\in(L\times L)}$$

We generate the analogue of initial state probabilities as the one-hot encoding of the first label state, or 0 if not at the first position.

$$f_{\text{initial}} = [\mathbf{1}_{t=1}\mathbf{1}_{y_1=k}]_{k\in L}$$

Finally, a bias term is included as a constant 1, and then the bias, emission, initial state, and transition features are flattened and concatenated into a single vector of dimension $m = 1 + FL + L + L^2$, where F is the dimensionality of the note-based features and L is the number of label states. Note this is not a minimal feature set, as the labels form a discrete probability space which can be parameterized with $L-1$ parameters.

2.3. Label Engineering

One issue in any segmentation task is that positive labels (phrase boundaries) can be very sparse in the dataset. To mitigate this issue, the set of labels can be augmented with intermediate states that reflect a given sequence element's distance to the nearest phrase boundaries. One of the label engineering approaches used by Zhang & Xia (2020) is to linearly increment the label of each successive note in a phrase (their other method "exponential" assigns continuous labels and does not work with CRFs). Linear ascending labels allow the number of notes that have occurred since the last phrase boundary to be encoded in the labels. Since musical phrases tend to be of consistent lengths, this provides additional information to the model that positively correlates with the likelihood of a phrase boundary. However, linear ascending labels also increase the number of labels in the model to the maximum number of notes M per phrase over the entire dataset, which can significantly increase model size. In particular, M multiplies the number of model parameters by an order of M to M^2, depending on whether the linear-chain CRF includes parameters for all possible label transitions.

As an alternative to linear ascending labels, we propose a reversed approach called "cadence" labelling, which linearly decrements label stats towards the phrase boundary from a fixed value of k notes before the phrase boundary. This approach naturally models musical cadences, which are consistent patterns of pitch and duration that terminate phrases in most European melodies. We ignore rests so that the "cadence" labels correspond to the number of notes that are remaining before the phrase boundary.

3. Linear Chain Conditional Random Fields

Conditional Random Fields (CRFs) are a discriminative model for sequential data, in that they model the conditional probability $p(\boldsymbol{y}|\boldsymbol{x})$, where $\boldsymbol{y}$ is the latent variable or label sequence to predict, and $\boldsymbol{x}$ is the sequence of observed data.

Figure 1 illustrates the relationship between CRFs and other graphical models. In general, CRFs are any undirected graphical model (UGM) whose nodes represent a set of latent variables $\boldsymbol{y}$ which are conditional on a disjoint set of observed variables $\boldsymbol{x}$. CRFs are the discriminative equivalent of Hidden Markov Models (HMMs) or general directed graphical models (DGMs), in that they model the same classes of probability distributions, but optimize for the conditional likelihood $p(\boldsymbol{y} \mid \boldsymbol{x})$ instead of the joint likelihood $p(\boldsymbol{x}, \boldsymbol{y})$.

Much like the benefits which logistic regression has over Naive Bayes, CRFs are able to have more robust classification performance. This is because a generative model must model $p(\boldsymbol{x})$ and $p(\boldsymbol{y} \mid \boldsymbol{x})$ with the same parameters θ:

$$p_g(\boldsymbol{y}, \boldsymbol{x}; \theta) = p(\boldsymbol{y}; \theta)p(\boldsymbol{x} \mid \boldsymbol{y}; \theta)$$
$$\text{rewritten as } p(\boldsymbol{x}; \theta)p(\boldsymbol{y} \mid \boldsymbol{x}; \theta)$$

On the other hand, a discriminative model is free to select different parameters for $p(\boldsymbol{x})$ and $p(\boldsymbol{y} \mid \boldsymbol{x})$, so that:

$$p_d(\boldsymbol{y}, \boldsymbol{x}; \theta)p(\boldsymbol{x}; \theta')p(\boldsymbol{y} \mid \boldsymbol{x}; \theta)$$

In particular, this allows a discriminative model to fit $p(\boldsymbol{y} \mid \boldsymbol{x})$ to data independently of modelling $p(\boldsymbol{x})$, which generally improves classification performance (Minka, 2005).

In this paper, we focus on the case of the linear-chain CRFs, which assumes the first order Markov property that y_t is independent of $y_1, y_2, \ldots, y_{t-2}$ conditional on y_{t-1}. A data sequence is $\boldsymbol{x} := x_0, x_1, \ldots, x_n$, where $x_0 = \star$ is a special beginning token and n is the length of the given sequence. This sequence is input into the CRF via feature functions. Let $f_j(\boldsymbol{x}, i, y_i, y_{i-1})$ be a feature function representing some measurement on data, where i is the i-th position in the sequence $\boldsymbol{x}$ and y_i is the label of x_i. For example, f_j could be the jth component of the feature vector described in section 2.2 at the ith sequence position. A score function $s(\boldsymbol{y}, \boldsymbol{x})$ sums over feature functions so that

$$s(\boldsymbol{y}, \boldsymbol{x}) = \sum_{j=1}^{m}\sum_{i=1}^{n} \lambda_j f_j(\boldsymbol{x}, i, y_i, y_{i-1}) \tag{1}$$

where m is the number of features, n is the length of the

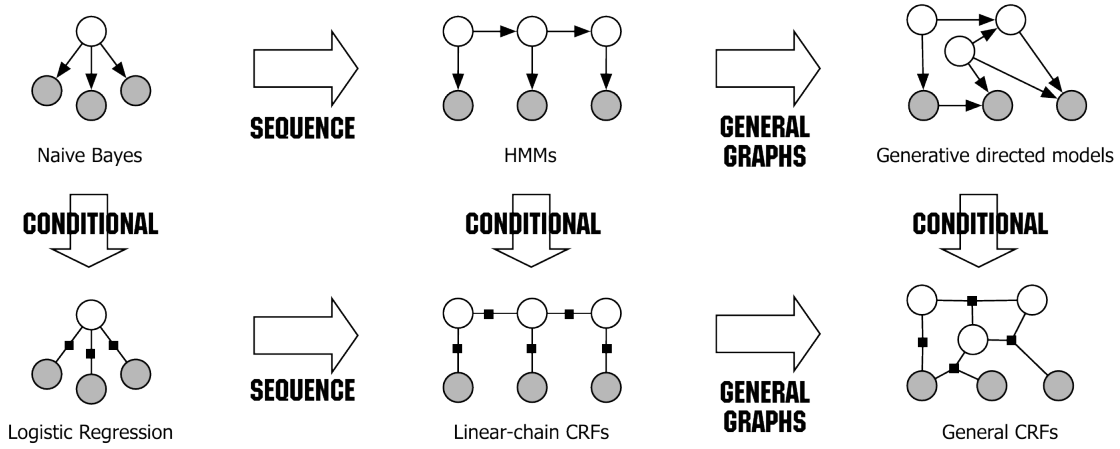

Figure 1. Comparison of directed and undirected graphical models of increasing flexibility of dependencies (left to right), and decreasing assumptions of the joint probability distribution (top to bottom). From Sutton & McCallum (2006).

sequence, and λ_j are the parameters of the CRF. The conditional probability that is modelled by the CRF is then

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp(s(\boldsymbol{y}, \boldsymbol{x}))}{\sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp(s(\boldsymbol{y}', \boldsymbol{x}))} \tag{2}$$

where $\mathcal{Y}$ is the label space containing all possible sequence labellings for the sequence.

The CRFs can also be thought of as a UGM which factorizes on edges. Let $\psi_{i,i-1}$ be the edge potential between labels y_i and y_{i-1}, and let Z be the normalizing constant for the distribution. Then

$$\psi_{i,i-1}(y_i, y_{i-1}) := \exp\left(\sum_{j=1}^{m} \lambda_j f_j(\boldsymbol{x}, i, y_i, y_{i-1})\right)$$

$$Z := \sum_{\boldsymbol{y}' \in \mathcal{Y}} \prod_{i=1}^{n} \psi_{i,i-1}(y'_i, y'_{i-1})$$

Using this formulation, the conditional probability in (2) can be rewritten as

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z} \prod_{i=1}^{n} \psi_{i,i-1}(y_i, y_{i-1}) \tag{3}$$

This definition clearly shows that a CRF is a UGM, and that like all UGMs, the CRF is in the exponential family of distributions. From this perspective, the parameters λ_j correspond to the natural parameters of the exponential family, and the feature functions f_j correspond to sufficient statistics. One problem which CRFs share with all UGMs is the intractability of computing the normalizing constant Z, which sums over a label space that can grow exponentially with sequence length. In the following section, we will show how the Markov assumption of the linear-chain CRF can reduce the time complexity of computing Z to make learning and inference feasible.

3.1. Learning, Inference, and Prediction

The CRF is trained by fitting the parameters λ_j to maximize the conditional probability from 2. Note that in the linear-chain CRF, the parameters λ_j are fixed for all sequences and timesteps, but this is not generally required. Let the training dataset be $\mathcal{T} = \{(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\}$, where $\boldsymbol{x}^{(k)}$ is a data sequence and $\boldsymbol{y}^{(k)}$ is the corresponding label sequence.

The parameters are fit according to their maximum likelihood estimators (MLE), which is equivalent to maximum entropy estimators in the exponential family. Estimation is conducted by maximizing the log-likelihood

$$\lambda^\star = \arg\max_{\lambda \in \mathbb{R}^m} \sum_k \log p(\boldsymbol{y}^{(k)}|\boldsymbol{x}^{(k)})$$

Unfortunately, no closed form solution exists for $\lambda^\star$ for CRFs. We therefure use the gradient ascent method to iteratively move towards the optimum parameters with the following update algorithm:

$$\lambda_j^{\text{new}} = t \frac{\partial \log p(\boldsymbol{y}|\boldsymbol{x})}{\partial \lambda_j} + \lambda_j^{\text{old}}$$

where t is a positive learning rate hyperparameter. The derivative of $\log p(\boldsymbol{y}|\boldsymbol{x})$ with respect to a particular λ_j is

$$\frac{\partial \log p(\boldsymbol{y}|\boldsymbol{x})}{\partial \lambda_j} = \sum_{i=1}^{n} f_j(\boldsymbol{x}, i, y_i, y_{i-1}) - \sum_{i=1}^{n} \sum_{\boldsymbol{y}' \in \mathcal{Y}} p(\boldsymbol{y}'|\boldsymbol{x}) f_j(\boldsymbol{x}, i, y_i, y_{i-1}) \tag{4}$$

As is typical of the exponential family, the derivative of the log-likelihood is the difference between the value of the feature functions given ground truth labels (empirical "moments"), and the expectation of the features under the

current model parameters (model "moments"). The model is optimized when its expectation is equal to the empirical feature values derived from data.

An issue with the above procedure is that, like the normalizing constant Z, calculating $p(\boldsymbol{y}' \mid \boldsymbol{x})$ requires summing over all possible label sequences. For a sequence of length n with L label states, the time complexity of this sum is L^n. However, the first order Markov assumption of the linear-chain CRF allows this sum to be expressed as the product of two recursions via the **forward-backward algorithm**, also called α-β recursion (Algorithm 1). This algorithm is typically used to fit HMMs, and can be applied to linear-chain CRFs without alteration. In particular, we first write the conditional probability of the labels as:

$$\begin{aligned}
&\sum_{i=1}^{n} \sum_{\boldsymbol{y}' \in \mathcal{Y}} p(\boldsymbol{y}'|\boldsymbol{x}) f_j(\boldsymbol{x}, i, y_i, y_{i-1}) \\
=&\sum_{i=1}^{n} \sum_{a \in \mathcal{S}, b \in \mathcal{S}} \sum_{\substack{\boldsymbol{y}' \in \mathcal{Y}, \\ y_i = b, y_{i-1} = a}} p(\boldsymbol{y}'|\boldsymbol{x}) f_j(\boldsymbol{x}, i, y_i = b, y_{i-1} = a) \\
=&\sum_{i=1}^{n} \sum_{a \in \mathcal{S}, b \in \mathcal{S}} f_j(\boldsymbol{x}, i, y_i = b, y_{i-1} = a) q_i(a, b)
\end{aligned}$$

where $q_i(a, b) = \sum_{\boldsymbol{y}' \in \mathcal{Y}, y_i = b, y_{i-1} = a} p(\boldsymbol{y}'|\boldsymbol{x})$. $q_i(a, b)$ can be interpreted as the unnormalized probability $p(\boldsymbol{y}'|\boldsymbol{x})$ in the current model, where $y_i = b$ and $y_{i-1} = a$.

To compute $q_i(a, b)$, we use dynamic programming to recursively calculate the following values for α and β:

$$\begin{aligned}
\alpha(i, s) &= \sum_{s'} \alpha(i-1, s') \psi_{i,i-1}(s, s') \\
\beta(i, s) &= \sum_{s'} \beta(i+1, s') \psi_{i+1,i}(s', s)
\end{aligned}$$

where $\alpha(1, s)$ is set to π_s, the initial probability of state s, and $\beta(n, s) = 1$. This algorithm is equivalent to message passing on UGMs, where the α recursion performs message passing from the start of the sequence, and the β recursion performs message passing from the end. Once α and β are obtained, $q_i(a, b)$ can be computed as

$$\begin{aligned}
Z &= \sum_{s} \alpha(n, s) \\
\mu_i(a, b) &= \alpha(i-1, a) \psi_{i,i-1}(b, a) \beta(i, b) \\
q_i(a, b) &= \frac{\mu_i(a, b)}{Z}
\end{aligned}$$

The forward-backward algorithm has quadratic time complexity $\mathcal{O}(n|\mathcal{S}|^2)$. Therefore, the time complexity of computing the derivative $\partial \log p(\boldsymbol{y}|\boldsymbol{x}) / \partial \lambda_j$ for each gradient ascent iteration is $\mathcal{O}(n|\mathcal{S}|^2)$.

Once the CRF is trained, it can be used to predict the label sequence $\boldsymbol{y}$ of a previously unseen sequence of observations $\boldsymbol{x}^{(new)}$. Prediction involves choosing the label sequence with the highest probability:

$$\boldsymbol{y}^{(predict)} = \arg\max_{\boldsymbol{y}' \in \mathcal{Y}} p(\boldsymbol{y}'|\boldsymbol{x}^{(new)})$$

The Viterbi algorithm (Algorithm 2), which is the forward-backward algorithm with sums replaced by argmax, can be used to compute $\boldsymbol{y}'$ in quadratic time.

4. Experiment

Our code for experiments is available online [2]. We use a ratio of 9:1 for the split of the training set and the testing set, and then report performance on the held-out testing set. We introduce the evaluation metrics, and baselines used for comparison in the following subsections.

4.1. Evaluation Metrics

The experimental results are evaluated via the precision, recall, and F-1 scores of the positive (1) labels corresponding to phrase beginnings. We only look at positive labels since underfitting models tend to label all notes with the 0 label due to the sparsity of the positive labels. Therefore, we report scores on the positive labels as they are a better indicator of performance.

To fairly evaluate models that are trained with different label engineering methods, we convert the predicted label sequences of all models to binary phrase start labels before conducting evaluation. In particular, to convert from linear ascending labels, we mark each of the first (smallest) labels as phrase starts. To convert from cadential labels, we mark the first non-rest note that follows the last (smallest) cadence label as a phrase start.

4.2. Baselines

We use the following baselines to evaluate our model implementation, features and labels. To ensure our implementation of the linear-chain CRF is functioning properly, we compare its performance with the library *sklearn_crfsuite*. We then compare performance on four hand-crafted features:

1. **Single Note:** This representation does not include edges in the PGM (the label of each note is independent of other notes).
2. **Numeric:** Pitches and durations are represented as continuous variables. Features based on music theory

[2] https://colab.research.google.com/drive/1lwthukErw0yrxIPcBWCJQeGoGJhzQMQF?usp=sharing

are also incorporated. The relative pitch and duration between the current note at t and the previous note at $t-1$, as well as the notes at $t-1$ and $t-2$, are included.

3. **Categorical:** Pitches and durations are represented as discrete categories. The pitch and duration of notes at t, $t-1$, and $t-2$ are included as standalone features.

4. **LSTM:** We use the output of a bidirectional LSTM neural network as the input features. Details about the LSTM are given in the next subsection.

We compare our proposed method of linearly decreasing "cadential" labels with the original binary labels as well as the linear ascending labels proposed by Zhang & Xia (2020).

4.3. LSTM Features

We generate features with a bidirectional LSTM for our CRF implementation as follows. Let P be the $n \times 2$-sized output dimensions of the LSTM neural network at each sequence position, where n is the number of features and 2 is the number of labels. $P_{i,j}$ denotes the score of the j^{th} label of the i^{th} note in a melodic phrase. Let A denote a transition matrix. $A_{i,j}$ denotes the transition score from the label i to label j. Then, the score function $s(\boldsymbol{y}, \boldsymbol{x})$ is represented as:

$$s(\boldsymbol{y}, \boldsymbol{x}) = \sum_{i=1}^{n} A_{y_{i-1}, y_i} + \sum_{i=0}^{n} P_{i, y_i} \tag{5}$$

Combining Equation 5 and Equation 2, we maximize the log-probability in Equation 6 during training, where $\boldsymbol{y_x}$ represents all possible label sequences. In the prediction part, we use Viterbi algorithm which is similar to that in the original CRF.

$$\log(p(\boldsymbol{y} \mid \boldsymbol{x})) = s(\boldsymbol{y}, \boldsymbol{x}) - \log\left(\sum_{\tilde{\boldsymbol{y}} \in \boldsymbol{y_x}} e^{s(\tilde{\boldsymbol{y}}, \boldsymbol{x})}\right) \tag{6}$$

We use the *PyTorch* library to construct our LSTM model. We set the embedding size to 128 and the hidden size to 128. We train with minibatch gradient descent for 15 epochs, using a batch size of 64 and the Adam optimizer (Kingma & Ba, 2015) with learning rate 0.001.

5. Results

5.1. CRF Comparsion

We first compare our implementation of the linear-chain CRF with the library *sklearn_crfsuite*, using *Numeric* features and binary labels. Table 1 presents the experimental results. From this table, we can see that our implementation of CRF can complete the melody segmentation task, but the performance is lower than that of the library *sklearn*. This is likely because we have omitted L2 regularization from our implementation, which increases the chance of overfitting on the large number of features we use.

Model	Precision	Recall	F-1 score
CRF	81.78%	49.04%	61.31%
sklearn	87.52%	54.50%	67.17%

Table 1. Experimental results comparing our implementation of the linear-chain CRF and the library *sklearn_crfsuite*.

5.2. Features Comparison

Second, we compare different features while using the original binary labels. We show the experimental results in Table 2. As expected, modeling neighbor dependencies in the linear chain CRF improves overall performance (F-1 score) over *Single Notes*. However, none of the human-crafted features have competitive overall performance with LSTM features.

We notice that although LSTM features significantly improve overall performance, the LSTM has the lowest precision score. This indicates that the phrase starts identified by hand-crafted features are accurate, but many other positive label are missed. On the other hand, LSTM-identified phrase starts are less accurate but more thorough. Another possible explanation is that the precision of poorly-performing models is dominated by the first label in each sequence, which is guaranteed to be 1. As models improve in performance, a larger proportion of their predicted labels are in positions other than the initial position.

Feature Type	Precision	Recall	F-1 Score
Single Notes	100.00%	17.36%	29.58%
Numeric	86.30%	52.65%	65.40%
Categorical	83.80%	57.30%	68.06%
LSTM	79.98%	69.44%	74.33%

Table 2. Experimental results comparing the performance of hand-crafted and LSTM neural network features.

5.3. Labels Comparison

Finally, we compare different label engineering methods using the *Numeric* features. The experimental results are shown in Table 3. Our proposed cadence labelling method performs better than the original binary labels overall, but still worse than the linear ascending method. We again

observe a trend where the methods that perform better in F-1 score have reduced precision.

Label Type	Precision	Recall	F-1 Score
Binary	86.30%	52.65%	65.40%
Linear+Increase	74.60%	72.02%	73.29%
Linear+Decrease	80.47%	56.61%	66.46%

Table 3. Experimental results comparing different label engineering methods.

6. Conclusion

We have shown that a linear-chain CRF with hand-crafted features does not significantly improve performance on the melodic phrase segmentation task. Our hand-crafted features have F-1 scores comparable to those reported by Zhang & Xia (2020) for the best performing heuristic-based methods. We also see that LSTM features significantly improve the performance of our linear-chain CRF. We therefore conclude that the performance of the model in Zhang & Xia (2020) can be attributed mainly to the deep neural network models used to generate features, and not to the linear-chain CRF.

We propose some areas for future research into improving the performance of CRF models for phrase segmentation. The skip-chain CRF model described in (Sutton & McCallum, 2006) or generalized CRF models could be used to represent arbitrary undirected graphs. This would enable edges to be included between labels assigned to different melodies, allowing information from one melody to be used to predict labels in another. Additionally, edges could also be included between labels assigned to common musical n-grams. Musical n-grams tend to correlate well with musical phrases, since melodies tend to have many repeated patterns in terms of relative pitch or duration, especially at cadences. Including dependency between labels for a given n-gram would capture this similarity.

References

Cenkerová, Z., Hartmann, M., and Toiviainen, P. Crossing phrase boundaries in music. In *Proceedings of the Sound and Music Computing Conferences*, 2018.

Cenkerov´a, Z., Hartmann, M., and Toiviainen, P. Crossing phrase boundaries in music. *Proceedings of the Sound and Music Computing Conferences*, 2018.

Doornbusch, P. Gerhard nierhaus: Algorithmic composition: Paradigms of automated music generation. *Computer Music Journal*, 34:70–74, 09 2010.

Hashida, M., Matsui, T., and Katayose, H. A new music database describing deviation information of performance expressions. In *Proceedings of the 9th International Society for Music Information Retrieval Conference*, pp. 489–494, 2008.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

Minka, T. Discriminative models, not discriminative training. Technical report, Technical Report MSR-TR-2005-144, Microsoft Research, 2005.

Raphael, C. Automatic segmentation of acoustic musical signals using hidden markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):360–370, 1999.

Schaffrath, H. The Essen folksong collection in database containing 6,255 folksong transcriptions in the kern format and a 34-page research guide. 1995.

Sutton, C. and McCallum, A. An introduction to conditional random fields for relational learning. *Introduction to statistical relational learning*, 2:93–128, 2006.

Temperley, D. *The cognition of basic musical structures*. MIT press, 2004.

Wiggins, G. A. Computer representation of music in the research environment. In *Modern Methods for Musicology*, pp. 27–42. Routledge, 2016.

Zhang, Y. and Xia, G. Symbolic melody phrase segmentation using neural network with conditional random field. In *National Conference on Sound and Music Technology*, pp. 55–65. Springer, 2020.

7. Appendix: Algorithms

The following pseudo-code outlines the procedure of the forward-backward algorithm for computing the conditional probability of labels given a sequence of observations, as well as the procedure of the Viterbi decoding algorithm for computing the maximal probability label sequence given a sequence of observations.

Algorithm 1 Forward-backward algorithm

Input: length n, set of possible states $\mathcal{S}$, function $\psi_{i,i-1}(y_i, y_{i-1})$
Output: function α(i, s), function β(i, s)
for $\forall s \in \mathcal{S}$ **do**
  $\alpha(1, s) \leftarrow \psi_{1,0}(s, \star)$
end
for $\forall i \in \{2, \ldots, n\}; \forall s \in \mathcal{S}$ **do**
  $\alpha(i, s) \leftarrow \sum_{s' \in \mathcal{S}} \alpha(i-1, s')\psi_{i,i-1}(s, s')$
end
for $\forall s \in \mathcal{S}$ **do**
  $\beta(n, s) \leftarrow 1$
end
for $\forall i \in \{1, \ldots, n-1\}; \forall s \in \mathcal{S}$ **do**
  $\beta(i, s) \leftarrow \sum_{s' \in \mathcal{S}} \beta(i+1, s')\psi_{i+1,i}(s', s)$
end

Algorithm 2 Viterbi algorithm

Input: data sequence $\boldsymbol{x}^{(new)}$, length n, set of possible states $\mathcal{S}$, function $\psi_{i,i-1}(y_i, y_{i-1})$
Output: label sequence $\boldsymbol{y}^{(predict)}$
for $\forall s \in \mathcal{S}$ **do**
  $\alpha(1, s) \leftarrow \psi_{1,0}(s, \star)$
end
for $\forall i \in \{2, \ldots, n\}; \forall s \in \mathcal{S}$ **do**
  $\alpha(i, s) \leftarrow \sum_{s' \in \mathcal{S}} \alpha(i-1, s')\psi_{i,i-1}(s, s')$
  $a(i-1, s) \leftarrow \arg\max_{s' \in \mathcal{S}} \alpha(i-1, s')\psi_{i,i-1}(s, s')$
end
$b(n) \leftarrow \arg\max_{s \in \mathcal{S}} \alpha(n, s)$
for $\forall i \in \{1, \ldots, n-1\}$ **do**
  $b(i) \leftarrow a(i, b(i+1))$
end
$\boldsymbol{y}^{(predict)} \leftarrow concatenate(b(1), b(2), \ldots, b(n))$