Project in CSE 250B
Assignment 3: Conditional Random Fields

Andreas Landstad, Spencer Bliven, Jonas Hoelzler
Computer Science Department
University of California, San Diego
landstad.andreas@gmail.com, sbliven@ucsd.edu, jonas@hoelzler.de

Abstract

In this report we present our results from using Conditional Random Fields (CRF) to segment words in the Zulu language. A CRF model with different generic feature functions is used to segment novel Zulu words. The feature functions captured patterns in prefixes and suffixes, vowel/consonant patterns and patterns for windows of three to seven consecutive specific letters. The weights are learned using Collins perceptron Algorithm and Stochastic Gradient Descent (SGD). Our model obtained about 74% word accuracy and 96% letter level accuracy.

1. Introduction

Conditional random fields provide a generic model for performing supervised learning. Generic feature functions are used to transform complicated input data and labels into real numbers. This abstraction allows machine learning and statistical analysis on very complicated datasets.

One problem for which this is useful is automatic detection of syllable boundaries within words. Possible inputs include all words in a given language, plus the possibility of additional neologisms which may be coined after the training period. Developing a classifier to determine syllable boundaries is an important task for automatic hyphenation, speech synthesis, and linguistic analysis.

1.1. Conditional Random Field model

The goal of supervised classification is to predict a label, $\bar{y}$, from some input data, $\bar{x}$. Conditional random fields do not place any restrictions on the type of data and labels being learned. To allow this abstraction, conditional random fields introduces the concept of *feature functions* of the form $F(\bar{x}, \bar{y})$ which attempt to measure the compatibility of $\bar{x}$ with the label $\bar{y}$. A large number of such feature functions are calculated for each $(\bar{x}, \bar{y})$ pair, and the weighted sum of J feature functions is used to determine the probability that $\bar{y}$ is the correct label for $\bar{x}$, according to the equation

$$p(\bar{y} \mid \bar{x}; \mathbf{w}) = \frac{\exp\left(\sum_{j=1}^{J} w_j F_j(\bar{x}, \bar{y})\right)}{z(\bar{x}; \mathbf{w})} \tag{1}$$

where $z(\bar{x}; \mathbf{w})$ normalizes the total mass over all possible labels y' to one and is equal to

$$z(\bar{x}; \mathbf{w}) = \sum_{\bar{y}'} \exp\left(\sum_{j=1}^{J} w_j F_j(\bar{x}, \bar{y})\right). \tag{2}$$

To use this model for inference, one simply assigns the most probable label for a given $\bar{x}$,

$$\hat{y} = \arg\max_{\bar{y}} \, p(\bar{y} \mid \bar{x}; \mathbf{w}) \tag{3}$$

$$= \arg\max_{\bar{y}} \sum_{j=1}^{J} w_j F_j(\bar{x}, \bar{y}). \tag{4}$$

The weights vector $\mathbf{w}$ is learned according to the principle of maximum likelihood. Thus, for some set of training pairs $(\bar{x}_n, \bar{y}_n)$ for $n = \{1, 2, \ldots, N\}$, the objective function is to maximize the total log likelihood with respect to $\mathbf{w}$ over all points of

$$LCL(\mathbf{w}) = \sum_{n=1}^{N} \log\, p(\bar{y}_n \mid \bar{x}_n;\, \mathbf{w}) \tag{5}$$

$$= \sum_{n=1}^{N} \left(\sum_{j=1}^{J} w_j F_j(\bar{x}_n, \bar{y}_n) - \log z(\bar{x}_n; \mathbf{w}) \right). \tag{6}$$

During training, L_2 regularization is used such that most weights are near zero. This is important to prevent overfitting, since the number of features is often greater than the number of training pairs, resulting in an underspecified system. The regularized objective is

$$LCL^*(\mathbf{w}) = \sum_{n=1}^{N} \log p(\bar{y}_n \,|\, \bar{x}_n;\, \mathbf{w}) - \lambda N \parallel \mathbf{w} \parallel^2 \tag{7}$$

$$= \sum_{n=1}^{N} \left(\sum_{j=1}^{J} w_j F_j(\bar{x}_n, \bar{y}_n) - \log z(\bar{x}_n; \mathbf{w}) \right) - \lambda N \parallel \mathbf{w} \parallel^2 \tag{8}$$

where λ is the strength of regularization.

1.2. Linear Chain CRFs

Training a CRF means finding the weight vector w that gives the best possible prediction for equation (4) for each training example $\bar{x}$. For the general case, computing Eq. 4 takes exponential time to iterate over all possible values.

This problem can be circumvented by restricting the model to linear chain CRFs. This refers to CRFs with features which depend only on two consecutive tags. This allows all features $F_j(\bar{x}, \bar{y})$ to be decomposed into low-level feature functions

$$F_j(\bar{x}, \bar{y}) = \sum_{i=1}^{M_n} f_j(y_{i-1}, y_i, \bar{x}, i). \tag{9}$$

For linear chain CRFs, Eq 4 can be computed efficiently by the Viterbi algorithm. The g_i values are assumed to be given by preprocessing as a m by m Matrix:

$$g_i(y_{i-1}, y_i) = \sum_{j=1}^{J} w_j f_j(y_{i-1}, y_i, \bar{x}, i)) \tag{10}$$

Let v range over the range of tags. Define $U(k, v)$ to be the score of the best sequence of tags from position 1 to position k, where tag number k is required to equal v. This is a maximization over $k-1$ tags because tag number k is fixed to have value v. Formally,

$$U(k, v) = max_{y_1..y_k} \sum_{i=1}^{k-1} g_i(y_{i-1}, y_i) + g_i(y_{i-1}, v) \tag{11}$$

which can be implemented efficiently recursive as

$$U(k, v) = max_u[U(k-1, u) + g_k(u, v)] \tag{12}$$

Now, one can backtrack the path which gives the highest probability.

1.3. Training via Stochastic Gradient Ascent

The weight vector $\mathbf{w}$ that maximizes Equation 8 can be found using gradient ascent methods. The gradient of the LCL with respect to the jth component of $\mathbf{w}$ is

$$\frac{\partial}{\partial \mathbf{w}_j} LCL^* = \sum_{n=1}^{N} \frac{\partial}{\partial \mathbf{w}_j} \log p(y_n \,|\, \bar{x}_n;\, \mathbf{w}) - \lambda N \frac{\partial}{\partial \mathbf{w}_j} \parallel \mathbf{w} \parallel^2 \tag{13}$$

$$= \sum_{n=1}^{N} \left(F_j(\bar{x}_n, y_n) - \mathbb{E}_{\bar{y}' \,|\, \bar{x}_n} [F_j(\bar{x}_n, \bar{y}')] \right) - 2\lambda N \mathbf{w}_j. \tag{14}$$

The expectation in equation 14 can be calculated by summing over all possible labels $\bar{y}'$ according to

$$\mathbb{E}_{\bar{y}' \,|\, \bar{x}_n} [F_j(\bar{x}_n, \bar{y}')] = \sum_{\bar{y}'} p(\bar{y}' \,|\, \bar{x}_n;\, \mathbf{w})\, F_j(\bar{x}_n, \bar{y}'). \tag{15}$$

For general CRFs computing the expectation is computationally intensive since it requires summing over V_n^M possible length-M_n labels. For linear chain CRFs this can be solved in time $O(M_n V^2 J^2 + V^2 M_n)$ using the *forwards-backwards* algorithm, which is large but polynomial. The forwards-backwards algorithm involves computing two functions recursively. Let $\alpha(i, u)$ be the unnormalized probability of the tag sequence $y_1 y_2 \ldots y_i$ ending with $y_i = u$. This can be calculated as

$$\alpha(i, u) = \sum_{k=1}^{V} \alpha(i-1, v_k) \mathcal{M}_i(v_k, u) \tag{16}$$

for all tags v_k, where $\mathcal{M}_i(v_k, u)$ is the score

$$\mathcal{M}_i(v, u) = \exp \sum_{j=1}^{J} w_j\, f_j(v, u, \bar{x}, i). \tag{17}$$

The base case for the forwards vector is

$$\alpha(1, v) = \mathcal{M}_i(\text{BEGIN}, v) \tag{18}$$

The backwards vector, $\beta(u, i)$, is the unnormalized probability of the tag sequence $\bar{y}_i \bar{y}_{i+1} \ldots \bar{y}_M$. This is given as

$$\beta(u, i) = \sum_{k=1}^{V} \beta(v_k, i+1) \mathcal{M}_{i+1}(u, v_k) \tag{19}$$

with base case

$$\beta(v, n) = \mathcal{M}_{i+1}(u, \text{END}). \tag{20}$$

Together, the forwards and backwards algorithm can be used to calculate the partition function (Eq. 2) and conditional expectation (Eq. 15).

$$z(\bar{x}_n; \mathbf{w}) = \sum_{k=1}^{V} \alpha(M_n, v_k) \mathcal{M}_{M_n}(v_k, \text{END}) \tag{21}$$

$$= \sum_{k=1}^{V} \beta(v_k, 1) \mathcal{M}_1(\text{BEGIN}, v_k) \tag{22}$$

$$p(y_i = u|\bar{x}; \mathbf{w}) = \frac{\alpha(i,u)\beta(u,i)}{z(\bar{x}; \mathbf{w})} \quad (23)$$

$$p(y_{i-1} = u, y_i = v|\bar{x}; \mathbf{w}) = \frac{\alpha(i,u)\mathcal{M}_{i+1}\beta(v,i+1)}{z(\bar{x}; \mathbf{w})} \quad (24)$$

$$\mathbb{E}_{\bar{y}' \mid \bar{x}_n}\left[F_j(\bar{x}_n, \bar{y}')\right] = \sum_{i=1}^{M_n} \mathbb{E}_{\bar{y}' \mid \bar{x}_n}\left[f_j(y_{i-1}, y_i, \bar{x}_n, i)\right] \quad (25)$$

$$= \sum_{i=1}^{M_n}\sum_{k=1}^{V}\sum_{l=1}^{V}\big[p(y_{i-1} = v_k, y_i = v_l \mid \bar{x}_n;\, \mathbf{w}) \cdot f_j(y_{i-1}, y_i, \bar{x}_n, i)\big] \quad (26)$$

$$= \sum_{i=1}^{M_n}\sum_{k=1}^{V}\sum_{l=1}^{V}\Big[f_j(y_{i-1}, y_i, \bar{x}_n, i) \cdot \frac{\alpha(i,u)\mathcal{M}_{i+1}\beta(v,i+1)}{z(\bar{x}; \mathbf{w})}\Big] \quad (27)$$

Using the forwards-backwards algorithm to calculate the gradient of $LCL*$, standard stochastic gradient ascent can be used to find the maximum $\mathbf{w}$ with the update rule

$$w_j \leftarrow w_j + \lambda \frac{\partial}{\partial w_j} LCL^*(yb_n|\bar{x}_n; \mathbf{w}) \quad (28)$$

for each training pair $n = 1, 2, \dots, N$.

1.4. Collins Perceptron Algorithm

Although the forwards-backwards algorithm allows the computation of the gradient in polynomial time, this is still a significant computational challenge. Collins perceptron Algorithm attempts to improve on that time by approximating the gradient.

By placing all the probability mass on the most likely $\bar{y}$ value, i. e. by using the approximation $p(\bar{y}|\bar{x}; w) = I(y = \hat{y})$, where $\hat{y} = argmax_y p(\bar{y}|\bar{x}; w)$, the stochastic gradient update rule simplifies to

$$w_j := w_j + F_j(\bar{x}, \bar{y}) \quad (29)$$
$$w_j := w_j + F_j(\bar{x}, \hat{y}) \quad (30)$$

This rule is applied for every weight w_j, for a given training example $\langle \bar{x}, \bar{y} \rangle$. Given a training example $\bar{x}$, the label $\hat{y}$ can be thought of as an impostor compared to the true label y **??**.

2. Methods

2.1. Zulu Dataset

The Zulu dataset consists of 10,040 Zulu words with correct segmentations indicated. There are 107 words in the corpus that have more than one correct segmentation and are ignored, leaving 9933 words. The longest word had 24 letters, the longest syllabus consists of 16 letters.

2.2. Label encoding

To encode the syllable segmentation, two tags were used. For each letter in $word$, a tag 1 was assigned if the letter occurred on the end of a syllable, or tag 0 otherwise. The last letter of each word was assigned label 0 to allow differentiation of syllable boundaries from word termini. For instance, the word `akabezwa` (*"disobey"*) is segmented as `a'ka'bez'w'a` and has label `10100110`.

An additional encoding method was considered but not implemented due to time constraints. Letters could be tagged with their position within a syllable. This would lead to label `12123112` for `akabezwa`. However, this method was expected to have poorer performance than the `01` encoding due to the skewed distribution of labels (since short syllables are significantly more likely than long syllables).

2.3. Feature Selection

Three models were used for generating features: window features, prefix/suffix features, and consonant/vowel features. Together these models produced around 73000 specific features, which vary slightly depending on the training set used.

Window-features. We made generic features that included all different windows of three, four, five, six or seven letters that existed in the words of the dataset. These features captured the letters of this part of the word together with the last two tags for each window. Formally, for all k-letter substrings $x_{i-k+1}, x_{w-k+2}, \dots, x_i$ of $\bar{x}$ ending at position i within all training words, and corresponding to label u, v at position $i-1, i$, we generate a feature of the form

$$f_j(\bar{x}, y_{i-1}, y_i, i) = \prod_{w=1}^{k} I(x_{i-w} = x_{i-w}) \cdot I(y_{i-1} = u)I(y_i = v) \quad (31)$$

Thus for a word `abcd` with label `0010` two 3-window features would be generated; one testing $I(x_{i-2}, x_{i-1}, x_i =$ "abc"$)I(\bar{y}_{i-1}, \bar{y}_i =$ "01$''$), and one testing $I(x_{i-2}, x_{i-1}, x_i =$ "bcd"$)I(y_{i-1}y_i =$ "10").

Prefix and suffix features. In order to capture beginnings and endings we also made similar features for prefixes and suffixes. These features are windows of length one or two which must occur at either the beginning or end of the word. For instance, the 1-prefix for letter `a` and label `0` would be

$$f_j(\bar{x}, y_{i-1}, y_i, i) = I(i = 1)I(x_i = a)I(y_{i-1}y_i = B0) \quad (32)$$

where the special label `B` marks the beginning of the label. A 2-suffix for the letters `ka` and label `00` would be

$$f_j(\bar{x}, y_{i-1}, y_i, i) = I(i = n)I(x_{i-1}x_i = ka)I(y_{i-1}y_i = 00) \quad (33)$$

Vowel and consonant features. The final type of feature we looked at consists of vowel and consonant patterns. South African Linguist Linda Van Huyssteen notes that Zulu syllables are often ‘open’ syllables, meaning they tend to end with vowels [4]. In order to capture this we made features with consonants/vowels(cv) in two and two letters together with their labels. We thus has 16 features of the form:

$$\begin{aligned} f_j(y_{i-1}, y_i, \bar{x}, i) = &I(y_{i-1} = 0)I(y_i = 1) \\ &I(x_{i-1} \in \mathcal{C})I(x_i \in \mathcal{V}) \end{aligned} \quad (34)$$

for all combinations of tags $0, 1$ and letters in sets $\mathcal{C}, \mathcal{V}$ representing consonants and vowels.

The example above is the feature capturing the consonant-vowel syllable boundary pattern. Other common patterns include vowel-consonant syllable boundaries and intra-syllable consonant-vowel patterns.

2.4. Design of the Viterbi algorithm

The Viterbi algorithm was implemented by first creating a table of score values $U(k, v)$. The table was created recursively by using Equation 12. The score for the begin tags were added to $U(0, v)$. Before starting the backtracking, the score for the end tag were added to U(0,k). The Viterbi algorithm has complexity $O(n^2)$.

2.5. Design of Stochastic Gradient Ascent

The parameter λ was optimized based on a grid search with $\lambda = \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ on the first training fold. Through this process the value of $\lambda = 10^{-}3$ was shown to be optimal (see figure 1). Subsequent folds used the same parameter settings for computational efficiency. Twenty-five epochs of SGD were allowed for convergence.

As with the Viterbi algorithm, the forwards-backwards algorithm iteratively calculates tables of values for $\alpha(\cdot, \cdot)$. Features occurring at the beginning and end of labels were used to initialize the forwards and backwards tables.

Several methods were used to check the correctness of the forwards-backwards implementation. The partition function and log likelihood were computed combinatorially for a small test word for all possible labels. This was found to be numerically equal to the equivalent results generated by the forwards-backwards algorithm. Also, after training $\mathbf{w}$ by SGD, it was found that

$$\sum_{(\bar{x}, \bar{y}) \in T} F_j(\bar{x}, \bar{y}) \approx \sum_{(\bar{x}, \cdot) \in T} \mathbb{E}_{\bar{y} \mid \bar{x}} \left[F_j(\bar{x}, \bar{y}') \right] \quad (35)$$

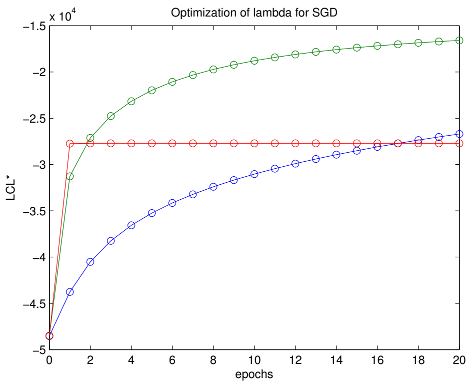

Figure 1. Convergence of SGD for various values of λ. The green curve ($\lambda = 10^{-3}$) represents the optimal value found by grid search. Higher values were also tried, but failed to converge for $\lambda > 10^{-3}$.

as predicted for the maximum LCL. Finally, gradients were checked numerically using the finite difference approximation.

2.6. Design of Collins Perceptron

The learning rate α was set to $0.02/T$. This is a widely used learning rate for gradient descent. Since Collins perceptron is similar to gradient descent, this learning rate seemed appropriate. The algorithm needed in our case only 3 epochs until convergence.

2.7. Evaluation

Five-fold cross validation was used to evaluate the performance of our algorithm. In each case, 7914 words were used to optimize parameter settings and train the feature weights. The remaining 1979 words were then used to test the accuracy of the trained classifier.

Two metrics were used to measure the accuracy of a prediction. For each test example, the Viterbi algorithm is used to predict a label, $\hat{y}_n$. The label is then compared to the true label, y_n, according to

1. Word Accuracy.

$$\frac{\sum_{n=1}^{N} I(\hat{y}_n = y_n)}{N} \quad (36)$$

2. Tag Accuracy

$$\frac{\sum_{n=1}^{N} \sum_{i=1}^{M_n} I(\hat{y}_{n,i} = y_{n,i})}{\sum_{n=1}^{N} M_n}. \quad (37)$$

3. Results

To test the algorithms the accuracy was measured using a 4-fold cross validation. Using Collins perceptron, the average word accuracy was about 74%. The letter level accuracy was about 96%. The results can be seen in Table 1.

	Word Level Accuracy	Letter Level Accuracy
Test 1	0.70	0.95
Test 2	0.80	0.97
Test 3	0.70	0.95
Test 4	0.77	0.96
Overall	0.74	0.96

Table 1. Prediction accuracy using Collins perceptron with the Viterbi algorithm

	Word Level accuracy	Letter Level Accuracy
Test 1	0.71	0.96
Test 2	0.72	0.96
Test 4	0.52	0.91
Overall	0.65	0.94

Table 2. Prediction Accuracy using SGD with the Viterbi algorithm. Test 3 was omitted.

	Weight	Feature
1	3.6157	Ends with a/0
2	-3.5913	VC/10
3	-3.1576	VC/01
4	-3.0107	CV/10
5	2.9385	Ends with e/0
6	2.8764	Ends with i/0
7	-2.8241	CC/10
8	-2.5115	CV/01
9	2.2844	Starts with ng/01
10	2.2588	Ends with o/0

Table 3. The top ten strongest weighted features. (Test 2)

The results using SGD are shown in Table 2. The overall results are slightly worse than using Collins perceptron; 65% of words are correct and 94% of tags. One explanation of this is that the perceptron algorithm is trained explicitly to maximize accuracy, while SGD is trying to maximize the LCL.

By looking at the relative weights one can estimate the relative contributions of various features towards predicting labels. Table 3 shows the top ten most strongly weighted features, while Figure 2 gives the full distribution of weights for one training set. Window features tend to have lower weights due to their large number, while the top ten weights are exclusively prefix, suffix, and consonant-vowel features.

4. Conclusion

Conditional Random Fields form a very general method for supervised learning. Their main strength comes from the huge variety of feature functions that can be used for analysis, and the fact that these features can be applied to non-numerical data such as natural languages.

In this study we were able to predict syllable boundaries with high accuracy. This can be attributed to our wide range of features, including both naive but comprehensive window features and more sophisticated consonant-vowel features based on prior linguistic knowledge.

While we were able to predict individual tags with high accuracy, 74% word accuracy may be too low for some applications, such as reliable hyphenation. However, this accuracy should be sufficient when supplemented by other methods. For example, a dictionary lookup could be used for most words, with a CRF classifier held in reserve for novel words.

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] L. Bottou. Stochastic gradient descent examples on toy problems, March 2011.

[3] C. Elkan. Log-linear models and conditional random fields, 2011.

[4] L. V. Huyssteen. *A Practical Approach to the Standardisation and Elaboration of Zulu as a Technical Language*. PhD thesis, University of South Africa, November 2003.

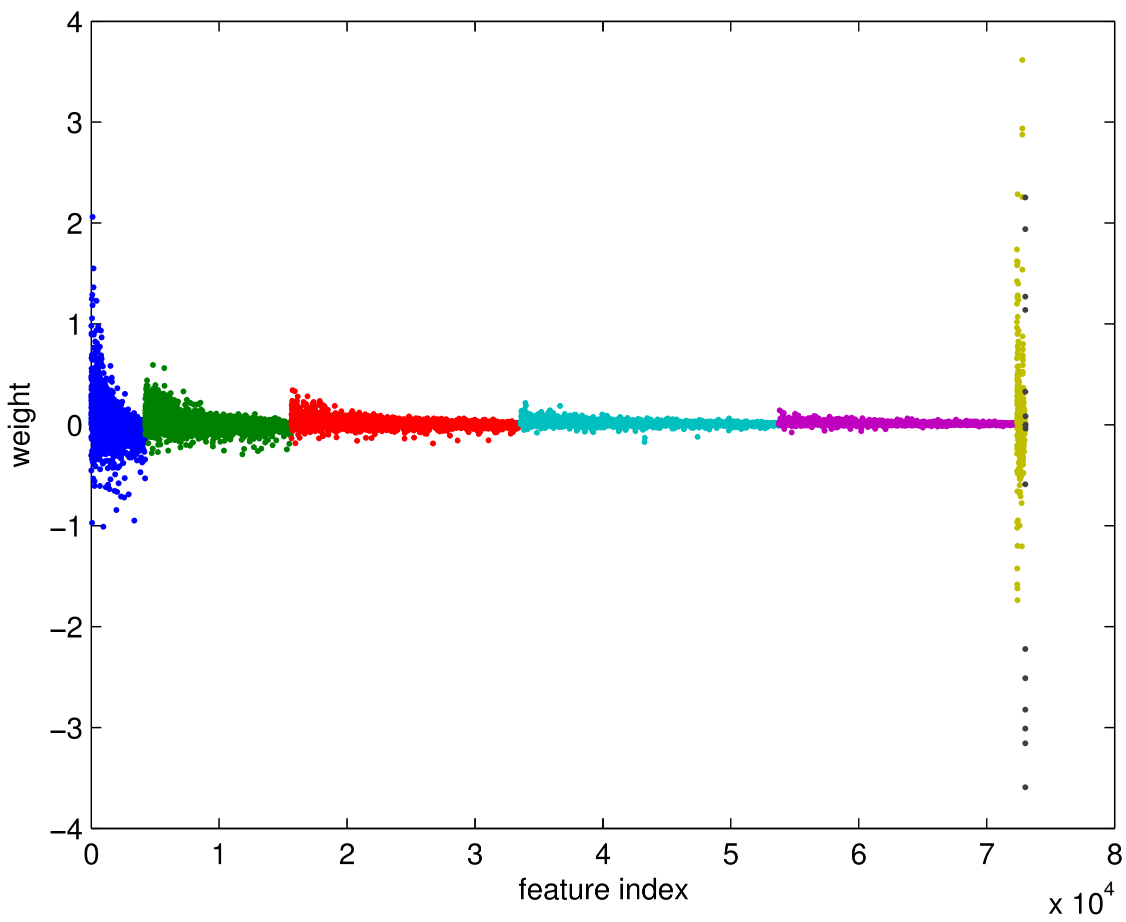

Figure 2. Feature weights (Test 2). Colors correspond to the type of feature: (blue–purple) 3–7 windows, (yellow) prefix and suffix, (black) consonant-vowel features