دانشگاه صنعتی شریف

مرکز زبان‌ها و زبان‌شناسی

پروژه درس زبان‌شناسی و ادبیات

استاد: دکتر بهروز محمودی بختیاری

سیستم تشخیص خودکار وزن عروضی اشعار کلاسیک فارسی

وحید مواجی

mavaji@alum.sharif.edu

گروه زبان‌شناسی رایانشی، دانشکده زبان‌ها و زبان‌شناسی، دانشگاه صنعتی شریف

چکیده

در این پروژه سعی بر آن داریم تا وزن عروضی اشعار کلاسیک فارسی را به طور خودکار تعیین کنیم. بدین منظور از یک سیستم تحلیل‌گر صرفی برای تبدیل متن فارسی به زنجیره واجی استفاده کرده و سپس این زنجیره واجی را طبق روش‌های موجود در عروض فارسی تقطیع کرده و وزن مناسب آن را می‌یابیم. از آنجا که خط فارسی فاقد مصوت‌ها است، لذا دقت آن ممکن است بسته به نمود واجی که پیشنهاد می‌دهد تغییر کند یا کاهش یابد. به همین دلیل این امکان در برنامه فراهم شده است که شعر را به صورت زنجیره واجی نیز وارد نموده و وزن آن را تعیین کنیم. در این کار از 31 وزن رایج و پرکاربرد زبان فارسی استفاده کرده‌ایم.

کلمات کلیدی: شعر کلاسیک، وزن، عروض، تشخیص خودکار وزن.

1. مقدمه

نیما می‌گوید: «وزن صدای احساسات و اندیشه‌های ماست. مردم ما صدا زودتر به ما نزدیکی می‌گیرند... هرکس اختیار خودش را دارد. ما در این جا تعزیه نگرفته‌ایم که قهرمانان واقعه همه‌شان منظوم با هم حرف بزنند، فقط مردم قبول نمی‌کنند و وزن می‌خواهند» *(نامه به شاملو)*.

اما مدت‌هاست که گروه بزرگی از شاعران دغدغه‌ای به نام وزن را رها کرده‌اند، چیزی که از نظر نیما یک ابزار ناگزیر برای شعر محسوب می‌شد و معتقد بود که در واقع وزن برای «بهتر متشکل‌کردن» شعر لازم است و سبب می‌شود که شاعر ارتباط عمیق‌تری با خواننده و مخاطب خود

برقرار سازد. از پیشینیان ابوعلی‌سینا آن را در خدمت «شورانندگی و خیال‌انگیزی» سخن می‌پنداشت و خواجه نصیر وزن را از آن سبب معتبر می‌دانست که به وجهی «اقتضاء خیال کند».

ما در این پروژه سعی بر این داریم تا با استفاده از فناوری رایانه و روش‌های زبان‌شناسی رایانشی، وزن شعر کلاسیک فارسی را تعیین کنیم. از آنجا که در شعر کلاسیک، معیار وزن مصراع می‌باشد، لذا ورودی این برنامه «مصراع» می‌باشد نه «بیت». لازم به ذکر است که در این نسخه اولیه برنامه، مواردی همچون «اختیارات شاعری»، «اختیارات وزنی»، و «عروض نیمایی» پشتیبانی نمی‌شود.

2. انواع هجا در زبان فارسی

هجاهای فارسی از نظر امتداد سه نوعند: هجای کوتاه، هجای بلند، هجای کشیده.

هجای کوتاه دارای دو واج است (صامت + مصوت کوتاه) مثل دَ، دِ، دُ، تَ، تِ، تُ. هجای کوتاه با علامت «U» نمایش داده می‌شود.

هجای بلند دارای سه واج است:

- صامت + مصوت کوتاه + صامت ← دَر، دِل، سُم
- صامت + مصوت بلند (هر مصوت بلند معادل دو واحد زمانی حساب می‌شود) ← با، بو، بی، سا، سو، سی.

هجای بلند با علامت «-» نمایش داده می‌شود.

هجای کشیده بیش از سه واج دارد و به یکی از صورت‌های زیر ظاهر می‌شود:

- صامت + مصوت کوتاه + صامت + صامت ← دَست، خُرد، رَنگ
- صامت + مصوت بلند + صامت ← بار، دیر، شوم
- صامت + مصوت بلند + صامت + صامت ← کاشت، ریخت، دوخت.

هجای کشیده با علامت «-U» نمایش داده می‌شود و از نظر امتداد معادل یک هجای بلند و یک هجای کوتاه است.

در تعیین وزن شعر فارسی اگر وزن یک مصرع (در شعر کلاسیک) یا یک سطر (در شعر نیمایی) را به‌دست آوریم، وزن شعر به‌دست آمده است. علت آن است که در زبان فارسی هم مثل بسیاری از زبان‌ها، واحد وزن شعر مصرع است. یعنی وقتی شاعر مصرع اول شعر را سرود وزن شعرش را انتخاب کرده است. در بقیه مصرع‌ها همان وزن باید تکرار شود. البته در عروض عربی واحد وزن بیت است و در عروض فارسی وقتی می‌خواهند اوزان شعر فارسی را نام‌گذاری کنند طبق سنت، بیت را واحد وزن می‌گیرند.

3. تقطیع هجایی

تقطیع یعنی تجزیه مصرع‌های شعر به هجاها، سپس تقسیم آن به ارکان تشکیل‌دهنده‌اش. منظور از تقطیع هجایی آن است که مصرع‌های مورد نظر را به هجاهای تشکیل‌دهنده آن تقسیم کنیم. برای این کار ابتدا مصرع‌های مورد نظر را به خط عروضی می‌نویسیم، سپس آن را به هجاهای تشکیل دهنده‌اش تقسیم می‌کنیم. مثال:

مبند ای دل به جز در یار خود دل امید از هر که داری جمله بگسل

مَبَندی دِل بِ جُز دَر یارِ خُد دِل اُمیدَز هَر کِ داری جُملِ بُگسِل

مَ|بَن|دی| دِل| بِ| جُز| دَر| یا|ا|ِ خُد| دِل اُ|می|دَز| هَر| کِ| دا|ری| جُم|لِ| بُگ|سِل

بعد از این که مصرع‌ها را به هجاهای تشکیل‌دهنده‌اش تقسیم کردیم، علامت آن‌ها را زیرشان می‌نویسیم:

مَ|بَن|دی| دِل| بِ| جُز| دَر| یا|ا|ِ خُد| دِل اُ|می|دَز| هَر| کِ| دا|ری| جُم|لِ| بُگ|سِل

\- - U - - - U - - - U - - U - - - U - - - U

4. تقطیع به ارکان

در تقطیع به ارکان، هجاهای مصرع‌ها را سه‌تا سه‌تا یا چهارتا چهارتا یا سه‌تا چهارتا یا چهارتا سه‌تا تقسیم می‌کنند. تقسیم‌بندی‌های دیگر در عروض امروز رایج نیست. علت آن است که بیشتر کلمات فارسی حداکثر چهار هجا دارند و کلماتی که بیش از چهار هجا دارند بسیار کم‌اند. همچنین تقسیم‌بندی چهارتا سه‌تا بسیار نامتداول است. مثال:

باور مکن که من دست از دامنت بدارم شمشیر نگسلاند پیوند مهربانان

باوَر مَکُن کِ مَن دَس تَز دامَنَت بِدارَم شَمشیر نَگسَلانَد پیوندِ مِهرَبانا

با | وَر | مَ | کُن | کِ | مَن | دَس | تَز | دا | مَ | نَت | بِ | دا | رَم

\- - U - U - - - - U - U - -

شَم | شیر | نَگ | سَ | لا | نَد | پی | ون | دِ | مِه | رَ | با | نا

\- - U - U - - - - U - U | - -

حال می‌توان به جای هجاهای کوتاه و بلند از «تَ» و «تَن» استفاده کنیم. مثلاً به جای اینکه بگوییم وزن شعر از چهاربار تکرار یک هجای کوتاه و سه هجای بلند [U---] تشکیل شده، می‌توان گفت: وزن آن شعر از چهار بار تکرار «تَ تَن تَن تَن» تشکیل شده است. ولی چون در صرف و نحو عربی همه کلمات را با «ف، ع، ل» می‌ستجند، در عروض عربی و فارسی هم وزن هجاهای تقطیع شده هر مصرع را (که نمایانگر نظم وزن‌اند) از «ف، ع، ل» ساخته‌اند. مثلاً هموزن (U--) فَعولن است. همچنین به جای (U---) از قالب هموزنش «مفاعیلن استفاده می‌کنند». این قالب‌ها را ارکان عروضی می‌نامند. مهمترین قالب‌های عروضی را وحیدیان کامکار نوزده‌تا دانسته است.

5. پیاده‌سازی

با توجه به مبانی نظری که به طور خلاصه در بخش‌های قبل ذکر شد، برنامه‌ای طراحی گردید که متن ورودی را گرفته و مراحل تقطیع هجایی و تقطیع به ارکان آن را انجام می‌دهد. البته اگر متن ورودی به خط فارسی باشد، ابتدا با الگوریتم‌هایی که برای آن طراحی شده است، یک (یا چند) نمود واجی برای آن پیدا می‌شود. با توجه به الگوی واجی زبان فارسی که به صورت‌های CV, CVC, CVCC است، هر گاه مصوتی پیدا شود، صامت قبل از آن متعلق به آن بوده و چون خوشه‌های همخوانی در آغازه هجا نداریم، لذا صامت‌های قبل از آن متعلق به هجای قبلی خواهند بود. سپس با استفاده از الگوریتم لونشتاین که الگوریتمی برای یافتن حداقل فاصله بین دو رشته است، در بین گروه‌های وزنی که در جدول 5-1 آمده‌اند، آن گروهی را که کمترین فاصله را با تقطیع متن ما دارد انتخاب کرده و به عنوان وزن شعر (مصرع) برمی‌گردانیم. الگورتیم لونشتاین، الگوریتمی بسیار پرکاربرد است که بر پایه برنامه‌نویسی پویا عمل می‌کند و در یافتن کمینه تغییرات بین دو رشته و علی‌الخصوص زنجیره‌های واجی بسیار پرکاربرد است. گونه غنی‌شده این الگوریتم می‌تواند جزئیات تغییرات حاصله بین دو رشته هم را تشخیص دهد. یعنی اینکه چه عملگرهایی (درج، حذف،...) و به چه ترتیبی باید روی رشته منبع عمل کند تا رشته مقصد حاصل گردد.

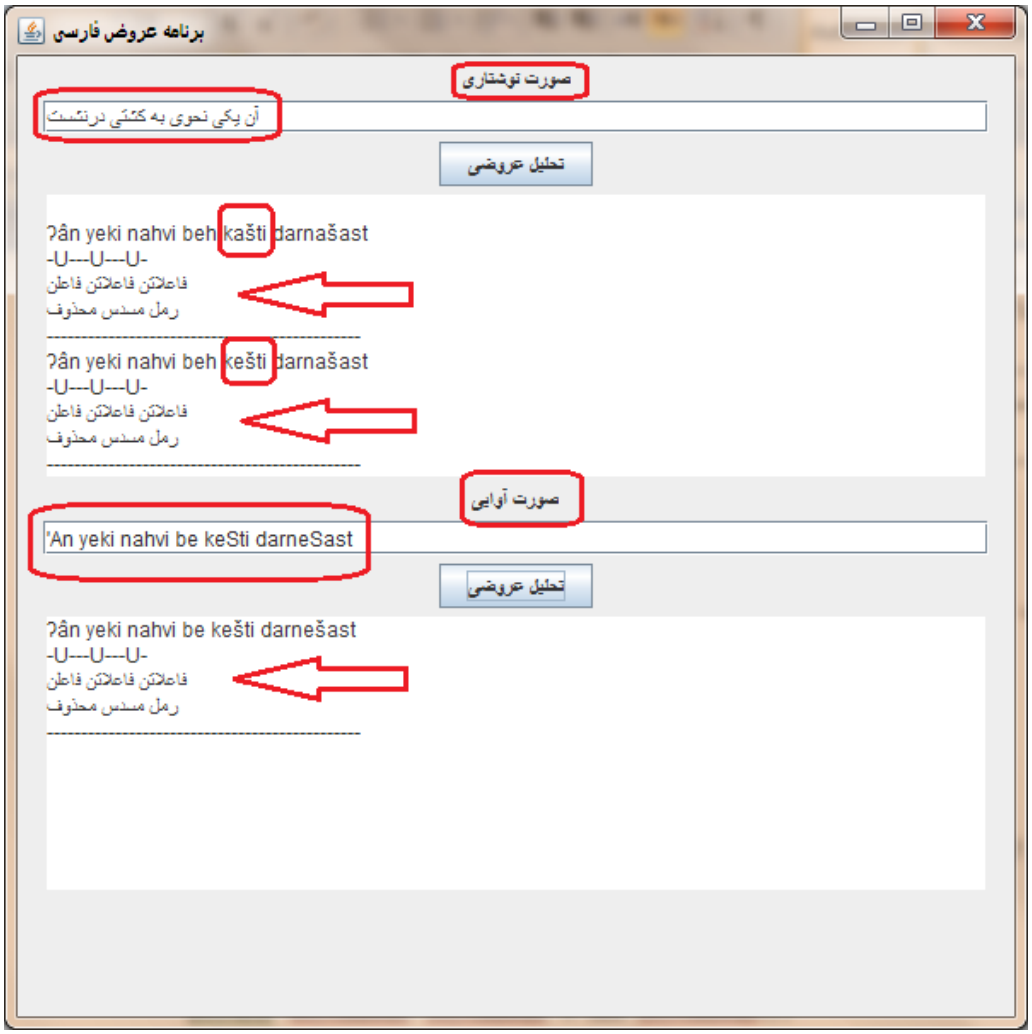
جدول 5-1 : وزن‌های پرکاربرد در زبان فارسی

تقطیع	گروه وزنی	نام گروه وزنی
UU--UU--UU--UU-	فعلاتن فعلاتن فعلاتن فعلن	رمل مثمن مخبون محذوف
U-U-UU--U-U-UU-	مفاعلن فعلاتن مفاعلن فعلن	مجتث مثمن محذوف
--U-U-UU--U-U-	مفعول فاعلات مفاعیل فاعلن	مضارع مثمن اخرب مکفوف محذوف

-U---U---U---U-	فاعلاتن فاعلاتن فاعلاتن فاعلن	رمل مثمن محذوف
U---U---U---U--U	مفاعيلن مفاعيلن مفاعيلن مفاعيل	هزج مثمن سالم
U---U---U--	مفاعيلن مفاعيلن فعولن	هزج مسدس محذوف
--U-U----U-U--	مفعول فاعلاتن مفعول فاعلاتن	مضارع مثمن اخرب
--UU--UU--UU--	مفعول مفاعيل مفاعيل فعولن	هزج مثمن اخرب مكفوف محذوف
UU--U-U-UU-	فعلاتن مفاعلن فعلن	خفيف مخبون محذوف
-U---U---U-	فاعلاتن فاعلاتن فاعلن	رمل مسدس محذوف
-UU-U-U--UU-U-U-	مفتعلن مفاعلن مفتعلن مفاعلن	رجز مثمن مطوي مخبون
--UU-----UU---	مفعول مفاعيلن مفعول مفاعيلن	هزج مثمن اخرب
UU-U-U--UU-U-U--	فعلات فاعلاتن فعلات فاعلاتن	رمل مثمن مشكول
U-U-UU--U-U-UU--	مفاعلن فعلاتن مفاعلن فعلاتن	مجتث مثمن مخبون
-UU-U-U--	مفعول مفاعلن فعولن	هزج مسدس اخرب مقبوض محذوف
--U----U--	مستفعلاتن مستفعلاتن	متقارب مثمن اثلم
-UU--U-U-UU--	مفتعلن فاعلات مفتعلن فع	منسرح مثمن مطوي منحور
--U---U---U---U-	مستفعلن مستفعلن مستفعلن مستفعلن	رجز مثمن سالم
-UU--U--UU--U-	مفتعلن فاعلن مفتعلن فاعلن	منسرح مثمن مطوي مكشوف
-U-U----U-U---	فاعلات مفعولن فاعلات مفعولن	مقتضب مثمن مطوي مقطوع
-UU--UU--U-	مفتعلن مفتعلن فاعلن	سريع مطوي مكشوف
U--U--U--U--	فعولن فعولن فعولن فعولن	متقارب مثمن سالم
U--U--U--U-	فعولن فعولن فعولن فعل	متقارب مثمن محذوف
-UU--UU--UU--UU-	مفتعلن مفتعلن مفتعلن مفتعلن	رجز مثمن مطوي
UU--UU--UU--UU--	فعلاتن فعلاتن فعلاتن فعلاتن	رمل مثمن مخبون
UU--UU--UU-	فعلاتن فعلاتن فعلن	رمل مسدس مخبون محذوف
U--UU--UU--UU--	مفاعيل مفاعيل مفاعيل فعولن	هزج مثمن مكفوف محذوف
UU--U-U-UU--U-U-	فعلاتن مفاعلن فعلاتن مفاعلن	خفيف مثمن مخبون

--UU-U-U---	مفعول مفاعلن مفاعیلن	هزج مسدس اخرب مقبوض
-U---U---U---U--	فاعلاتن فاعلاتن فاعلاتن فاعلاتن	رمل مثمن سالم
--U-UU---U-UU-	مستفعلن فعلن مستفعلن فعلن	بسیط مثمن مخبون

در شکل 5-1 نمونه‌ای از خروجی برنامه را هم به صورت ورودی خط فارسی و هم ورودی واج‌نویسی شده می‌بینیم.



شکل 5-1: خروجی نمونه برنامه برای مصرع «آن یکی نحوی به کشتی درنشست»

متن قسمت اصلی برنامه تشخیص وزن عروضی به شرح ذیل می‌باشد:

```
public class ProsodyAnalyzer {
    private PhonemeAnalyzer phonemeAnalyzer = new PhonemeAnalyzer();
    private char[] shortVowels = new char[]{'a', 'e', 'o'};
    private char[] longVowels = new char[]{'â', 'i', 'u'};
    private RhythmBean rhythmBean = new RhythmBean();

    private Rhythm[] rhythms;

    public ProsodyAnalyzer() {
```

```
        init();
    }

    public void init() {
        rhythms = rhythmBean.getAllRhythms();
    }

    public PhonemeHolder[] getProsody(String poem, boolean phoneme) {
        PhonemeHolder[] phonemeHolders = segment(poem, phoneme);

        PhonemeHolder[] results = new PhonemeHolder[0];
        for (PhonemeHolder phonemeHolder : phonemeHolders) {
            int min = Integer.MAX_VALUE;
            int index = 0;

            for (int i = 0; i < rhythms.length; i++) {
                int distance = getLevenshteinDistance(phonemeHolder.getSegment(), rhythms[i].getSymbol());
                if (distance < min) {
                    min = distance;
                    index = i;
                }
            }

            results = (PhonemeHolder[]) ArrayUtils.add(results,
                    new PhonemeHolder(phonemeHolder.getPhonologicalForm(),
                            phonemeHolder.getSegment(), rhythms[index]));
        }

        return results;
    }

    public PhonemeHolder[] segment(String poem, boolean phoneme) {
        String[] phonologicalForms;
        if (phoneme) {
            phonologicalForms = new String[]{poem};
        } else {
            phonologicalForms = phonemeAnalyzer.textToPhoneme(poem);
        }

        PhonemeHolder[] phonemeHolders = new PhonemeHolder[0];
        for (String phonologicalForm : phonologicalForms) {

            String result = "";
            String[] syllables = syllables(phonologicalForm);
            for (int i = 1; i < syllables.length; i++) {
                result = getRhythmicSyllable(syllables[i]) + result;
            }
//            segments = (String[]) ArrayUtils.add(segments, result + "-");
            phonemeHolders       =       (PhonemeHolder[])       ArrayUtils.add(phonemeHolders,       new
PhonemeHolder(phonologicalForm, result));
        }
        return phonemeHolders;
    }

    public String[] syllables(String phonologicalForm) {
        phonologicalForm = phonologicalForm.replaceAll(" ", "");
        if           (ArrayUtils.contains(shortVowels,           phonologicalForm.charAt(0))           ||
ArrayUtils.contains(longVowels, phonologicalForm.charAt(0))) {
            phonologicalForm = "?" + phonologicalForm;
        }

        String[] syllables = new String[0];
        for (int i = phonologicalForm.length() - 1; i >= 1; i--) {
            char c = phonologicalForm.charAt(i);

            if (ArrayUtils.contains(shortVowels, c) || ArrayUtils.contains(longVowels, c)) {
                String syllable = phonologicalForm.substring(i - 1);
                syllables = (String[]) ArrayUtils.add(syllables, syllable);

                if (i >= 1) {
                    phonologicalForm = phonologicalForm.substring(0, i - 1);
                    i = i - 1;
                }
            }
        }

        return syllables;
    }
```

```
public String getSyllable(String s) {
    String syllable = "C";

    if (ArrayUtils.contains(shortVowels, s.charAt(1))) {
        syllable += "V";
    } else {
        syllable += "L";
    }

    for (int i = 2; i < s.length(); i++) {
        syllable += "C";
    }

    return syllable;
}

public String getRhythmicSyllable(String s) {
    String result = "";

    String syllable = getSyllable(s);

    if (syllable.equals("CV")) {
        result = "U";
    } else if (syllable.equals("CL") || syllable.equals("CVC")) {
        result = "-";
    } else if (syllable.equals("CVCC") || syllable.equals("CLC") || syllable.equals("CLCC")) {
        result = "-U";
    }

    return result;
}

public static int getLevenshteinDistance(String s, String t) {
    if (s == null || t == null) {
        throw new IllegalArgumentException("Strings must not be null");
    }

    /*
      The difference between this impl. and the previous is that, rather
       than creating and retaining a matrix of size s.length()+1 by t.length()+1,
       we maintain two single-dimensional arrays of length s.length()+1.  The first, d,
       is the 'current working' distance array that maintains the newest distance cost
       counts as we iterate through the characters of String s.  Each time we increment
       the index of String t we are comparing, d is copied to p, the second int[].  Doing so
       allows us to retain the previous cost counts as required by the algorithm (taking
       the minimum of the cost count to the left, up one, and diagonally up and to the left
       of the current cost count being calculated).  (Note that the arrays aren't really
       copied anymore, just switched...this is clearly much better than cloning an array
       or doing a System.arraycopy() each time  through the outer loop.)

       Effectively, the difference between the two implementations is this one does not
       cause an out of memory condition when calculating the LD over two very large strings.

    */

    int n = s.length(); // length of s
    int m = t.length(); // length of t

    if (n == 0) {
        return m;
    } else if (m == 0) {
        return n;
    }

    int p[] = new int[n + 1]; //'previous' cost array, horizontally
    int d[] = new int[n + 1]; // cost array, horizontally
    int _d[]; //placeholder to assist in swapping p and d

    // indexes into strings s and t
    int i; // iterates through s
    int j; // iterates through t

    char t_j; // jth character of t

    int cost; // cost

    for (i = 0; i <= n; i++) {
```

```
            p[i] = i;
        }

        for (j = 1; j <= m; j++) {
            t_j = t.charAt(j - 1);
            d[0] = j;

            for (i = 1; i <= n; i++) {
                cost = s.charAt(i - 1) == t_j ? 0 : 1;
                // minimum of cell to the left+1, to the top+1, diagonally left and up +cost

                d[i] = Math.min(Math.min(d[i - 1] + 1, p[i] + 1), p[i - 1] + cost);
            }

            // copy current distance counts to 'previous row' distance counts
            _d = p;
            p = d;
            d = _d;
        }

        // our last action in the above loop was to switch d and p, so p now
        // actually has the most recent cost counts
        return p[n];
    }
}
```

6. منابع

[1] سلطانی طارمی، س، **عروض به زبان امروز**، نشر نی، 1390.

[2] شمیسا، س، **آشنايي با عروض و قافيه**، ویراست چهارم، نشر میترا، 1383.

[3] وحیدیان کامیار، ت، **فنون و صنايع ادبي(عروض)**، نشر ایران، 1365.

[4] ماهیار، ع، **عروض فارسي شيوه اي نو براي آموزش عروض و قافيه**، نشر قطره، 1374.

[5] تجلیل، ج، **عروض**، نشر همراه، پاییز 1368.