REVISTA SIGNOS. ESTUDIOS DE LINGÜÍSTICA ISSN 0718-0934
© 2017 PUCV, Chile DOI: 10.4067/S0718-09342017000300385 50(95) 385-407

Diseño e implementación de un corrector ortográfico dinámico para el sistema tutorial inteligente, ELE-TUTORA*

Design and implementation of a dynamic spell checker for the intelligent tutoring system, ELE-TUTORA

Anita Ferreira Cabrera
UNIVERSIDAD DE CONCEPCIÓN
CHILE
aferreir@udec.cl

Sergio Hernández Osuna
UNIVERSIDAD DE CONCEPCIÓN
CHILE
sergihernandez@udec.cl

Recibido: 16-V-2016 / **Aceptado:** 10-IV-2017

Resumen

El presente trabajo se enfoca en la problemática de la precisión lingüística en la producción escrita, específicamente en los errores ortográficos. Para ello, se diseñó y construyó un corrector ortográfico para un Sistema Tutorial Inteligente (STI) del Español como Lengua Extranjera, ELE-TUTORA[1]. Este corrector está programado en *Python* 3 y su diseño corresponde a una arquitectura dinámica, con el fin de evitar su rigidización futura y permitir mejorar su funcionamiento. En este artículo, se presenta la descripción de sus módulos, la implementación del código utilizado para programar el corrector, y se explica su funcionamiento a partir de salidas y ejemplos en la corrección de errores en el Español como Lengua Extranjera (ELE).

Palabras Clave: Precisión lingüística, ortografía, corrector ortográfico, python, sistema tutorial inteligente, Español como Lengua Extranjera, ELE.

Abstract

This article focuses on the problem of linguistic accuracy in written production, specifically misspellings. To this end, we designed and built a spell checker for an Intelligent Tutorial System (ITS) of Spanish as a Foreign Language, ELE-TUTORA. This corrector is programmed in Python 3 and its design corresponds to a dynamic architecture, so as to avoid its future rigidity and allow better performance. In this article, we present the description of its modules, the implementation of the code used to program the corrector, and its operation is explained from its output and examples in correcting errors in Spanish as a Foreign Language (SFL).

Key Words: Linguistic accuracy, spelling, spell checker, python, intelligent tutoring system. Spanish as a Foreign Language.

INTRODUCCIÓN

La precisión gramatical en el ámbito de la destreza escrita en Español como Lengua Extranjera (ELE) requiere de apoyos metodológicos innovadores que faciliten el mejoramiento de las problemáticas lingüísticas que tanto preocupan a profesores como estudiantes de Adquisición de Segundas Lenguas. Para ello es relevante innovar en el reconocimiento, análisis y tratamiento de los errores en ELE (De Alba Quiñonez, 2009). Los errores, entendidos como desviaciones de las normas de la lengua meta, son parte natural del proceso de adquisición y aprendizaje de una segunda lengua (Ellis, 1997; Alexopoulou, 2005; Vásquez, 2009). Nuestro enfoque de investigación se fundamenta en la necesidad de tratar el error a lo largo del proceso de adquisición y aprendizaje de una lengua extranjera (LE), porque este forma parte del mismo y, por lo tanto, es una evidencia de este proceso.

El Análisis de Errores, basado en los procedimientos del Corpus de aprendices (CLC, del inglés *Computer Learner Corpora*) y en el Análisis de Errores asistido por Computador (CEA, del inglés *Computer aided Error Analysis*) en lo que se refiere a Corpora de Aprendientes de ELE en Formato Electrónico, ha evidenciado que los errores de mayor frecuencia y recurrencia corresponden a los errores ortográficos (Ferreira, 2014a; Ferreira, Elejalde & Vine, 2014). Los estudios se han sustentado en el corpus CAELE (Ferreira, 2015) constituido por una colección de 418 textos de aprendientes de ELE, almacenados y procesados en formato digital. Estos textos han sido recolectados durante intervenciones lingüísticas entre los años 2014 y 2015 con el objeto de describir la interlengua de los aprendices e identificar los errores lingüísticos más frecuentes y recurrentes según el nivel de competencia (*proficiency*, en inglés, es decir la capacidad que una persona demuestra en el uso de una lengua extranjera). Los sujetos que participaron fueron 62 estudiantes extranjeros. Los resultados muestran un total de 4663 errores recurrentes, es decir, aquellos errores que se repiten en dos o más textos producidos por un sujeto, siendo la ortografía acentual, diacrítica y dierética, uno de los tipos más recurrentes con 2412 errores (57%) (Ferreira, 2014a).

Estos hallazgos han tenido implicaciones para la selección, identificación y tratamiento de los errores ortográficos procesados a través de un Sistema Tutorial Inteligente para el Español como Lengua Extranjera, el cual se está construyendo en el contexto de investigaciones Fondecyt[2]. Esto nos ha permitido primeramente apoyar el proceso de reconocimiento y tratamiento de errores gramaticales mediante técnicas de procesamiento de lenguaje natural, las cuales posibilitan el reconocimiento de los errores (*parser*) y la entrega de una ayuda focalizada y efectiva (*feedback* correctivo escrito). De esta forma, los estudiantes pueden resolver de manera autónoma sus errores de lengua y mejorar su precisión lingüística en la escritura (Ferreira & Kotz, 2010; Ferreira, Salcedo, Kotz & Barrientos, 2012). Recientemente, se ha estado trabajando en el ámbito del reconocimiento de los errores ortográficos basados en los

avances de los correctores ortográficos y en las técnicas y herramientas de procesamientos del lenguaje natural.

Ahora bien, de acuerdo con el Marco Común Europeo de Referencia, MCER (Consejo de Europa, 2002) la competencia ortográfica supone el conocimiento y la destreza en la percepción y la producción de los símbolos de que se componen los textos escritos, esto es, las formas de las letras en sus modalidades normal y cursiva, tanto mayúsculas como minúsculas, la correcta ortografía de las palabras, incluidas las contracciones, los signos de puntuación y sus normas de uso, las convenciones tipográficas y las variedades de tipos de letras.

Los correctores ortográficos, incluidos en los procesadores de textos, han facilitado enormemente la tarea para quienes producen escritos. Dichos correctores ortográficos, por lo general, contrastan las palabras con un patrón y sugieren realizar algún tipo de cambio en base a esto. Como ejemplo de lo anterior, si se escribiera la oración 'el gato salio' de 'la casa', el corrector ortográfico del procesador de textos de inmediato marcará el verbo 'salio' como mal escrito y sugerirá, probablemente, la forma 'salió'. Pero, por otra parte, si la oración fuera 'el perro esta comiendo', el procesador de textos no encontrará problema ortográfico alguno, ya que la palabra 'esta' existe entre sus patrones; sin embargo, dicha palabra ni siquiera es un verbo como requiere la oración escrita. He ahí un problema y una limitación que se produce con los correctores ortográficos al no ser capaces de discriminar entre palabras que están entre los modelos de su base de datos y las que no están. Pese a esto, hay que reconocer que el funcionamiento de estos correctores ha avanzado y que una aplicación como *MS Word* 2016, si bien no reconoce la palabra del ejemplo –'esta'– como error (subrayándola en rojo), sí la destaca como un posible caso a revisar por el usuario (subrayándola en azul).

El principal objetivo en este artículo es contribuir con el diseño e implementación de un corrector ortográfico para el español que detecte los errores en forma inteligente, que automatice el proceso y que sea eficiente, con el fin de integrarlo al Sistema Tutorial Inteligente, ELE-TUTORA (Ferreira, 2014b). La perspectiva empleada para enfocar el problema es claramente multidisciplinar, como se verá en el desarrollo del artículo, ya que se aborda el problema desde un punto de vista lingüístico y, también, informático.

1. Marco teórico

1.1. El Sistema Tutorial Inteligente ELE Tutora y la retroalimentación correctiva de errores de escritura

Generalmente, la retroalimentación correctiva implementada en los sistemas computacionales para la enseñanza de lengua en el ámbito tecnológico se ha limitado a

simples mensajes de error, sobre la base de un enfoque de interacción 'te equivocaste-inténtalo de nuevo' (*wrong-try again*), el cual brinda poca o nula información metalingüística acerca de la naturaleza específica de los errores del estudiante. Para poder dar una retroalimentación centrada en el error se requiere que los programas reconozcan los errores de lengua presentes (gramaticales, léxicos, ortográficos, etc). Para ello, estos sistemas utilizan técnicas específicas de analizadores sintácticos (*parsing*) para analizar la respuesta de los estudiantes e identificar los errores que se presentan en los enunciados. Estas capacidades de Procesamiento de Lenguaje Natural han permitido a los sistemas producir algunas estrategias de *feedback* más sofisticadas como las claves metalingüísticas y los 'informes de errores' basados en análisis de errores. A estos sistemas se les denomina 'inteligentes' atendiendo a la capacidad que ellos tienen para analizar gramaticalmente una entrada en lenguaje natural y luego producir un mensaje o enunciado en lenguaje natural correspondiente a una estrategia de *feedback* correctivo adecuado para el error focalizado del estudiante (Ferreira, 2007; Ferreira, Moore & Mellish, 2007). Para ello, se basan en teorías gramaticales para procesar la entrada del estudiante con el objeto de reconocer y analizar los errores que se presentan. A continuación nos referiremos brevemente al STI ELE-Tutora.

El sistema Tutorial Inteligente ELE-Tutora (Ferreira, 2014b) es un sistema para la enseñanza del español como lengua extranjera que apoya el tratamiento de errores gramaticales en la etapa de revisión de un texto con el objetivo de mejorar la competencia lingüística en los estudiantes de ELE. Actualmente está en funcionamiento, integrado en el aula virtual del Programa de Español como Lengua Extranjera de la Universidad de Concepción. El acceso al aula es restringido a quienes participen de los cursos que se ofrecen. La pantalla de acceso a la plataforma se puede visitar desde http://ele.udec.cl/aulavirtual/?lang=es.

El planteamiento inicial de ELE-Tutora fue el modelo de sistema tutorial inteligente propuesto por Ferreira et al. (2007). Una de sus primeras implementaciones fue realizada por Ferreira y Atkinson (2009) en el contexto del diseño y construcción del generador del *feedback* correctivo. El desarrollo del *parser* o analizador sintáctico para errores gramaticales y sintácticos fue llevado a cabo por Ferreira y Kotz (2010). El *parser* considera una gramática de errores con *buggy rules* (reglas agramaticales), que permite anticipar y detectar los errores gramaticales. Posteriormente, la arquitectura del STI fue ampliada por Ferreira et al. (2012). El sistema se estructuró a partir de tres componentes básicos: (1) 'El módulo del dominio', que considera los principios y parámetros relativos al enfoque metodológico de enseñanza 'Focalizado en la Forma' (del inglés *focus on form*) en que se sustenta el desarrollo de contenidos, actividades, ejercicios y estrategias de *feedback* correctivo que el sistema provee al estudiante durante el proceso de enseñanza-aprendizaje del ELE. (2) 'El módulo del estudiante', que contiene información acerca del aprendiz de ELE (su conocimiento del dominio,

su nivel de competencia (*proficiency*), sus errores gramaticales más recurrentes, su estilo de aprendizaje, etc.). (3) 'El módulo del tutor', que considera las capacidades y estrategias de enseñanza que el sistema tutorial usa en su interacción con el alumno, esto es, reconoce e identifica los errores de lengua que el estudiante comete (analizador automático), provee estrategias de *feedback* correctivo efectivas (generador automático de *feedback*) y adecuadas al tipo de error, nivel de competencia (*proficiency*) y estilo de aprendizaje que el estudiante ha evidenciado durante la ejecución de las actividades comunicativas. Últimamente, se ha abordado la problemática del procesamiento de los errores ortográficos para apoyar el mejoramiento de la competencia lingüística en los aprendientes de ELE.

1.2. Los correctores ortográficos

El funcionamiento primario de un corrector ortográfico sigue aún sosteniéndose de manera importante en el uso de diccionarios, tal cual como era en sus primeras etapas. Sin embargo, han surgido otras formas de enfrentar la problemática en base a análisis estadísticos. En este aspecto es interesante destacar el trabajo de San Mateo (2016) que propone un algoritmo de corrección ortográfica, para textos escritos en español, destinado a hablantes nativos competentes que realizan labores de corrección de textos. Los posibles errores se identifican por medio de análisis estadísticos, comparando parejas de palabras (bigramas) con las combinaciones utilizadas en un corpus de referencia de cien millones de vocablos. De esa forma se pueden detectar errores poco frecuentes, con la limitación de que no es posible identificar errores que no puedan ser colegidos del análisis de palabras adyacentes.

A diferencia de lo recién presentado, nuestra propuesta se enfoca en el uso de diccionarios. Tapia y Ariza (1997) señalaban que el corrector ortográfico requiere como punto de partida dos bases de datos: el diccionario principal y el diccionario secundario o diccionario del usuario. El primer diccionario viene a coincidir con una selección o versión abreviada de un diccionario académico del español, por ejemplo, el Diccionario de la Real Academia Española. El segundo diccionario se forma con las palabras que no figuran en el diccionario principal y que cada usuario va introduciendo en función de sus necesidades o conveniencias. El corrector, según los citados autores, cotejaba las palabras encontradas en el texto escrito con las contenidas en los diccionarios. En el fondo operaba buscando coincidencias de patrones (*pattern matching*, en inglés). Si el corrector encontraba la palabra del texto en alguno de los diccionarios, la consideraba correcta; de lo contrario, la consideraba incorrecta y, generalmente, ofrecía al usuario una alternativa de palabra entre otras cuya grafía fuera parecida. Esto último es algo que a cualquier usuario de una aplicación de mensajería como *WhatsApp*, por ejemplo, podrá parecerle conocido, y es que casi 20 años después del artículo citado más arriba, el funcionamiento de un corrector ortográfico,

si bien ha sufrido mejoras que refinan su precisión, esencialmente sigue operando de manera similar a la descrita.

En 2008 Plüss y Pomponio indicaban sobre los correctores ortográficos que el enfoque, aparentemente universal, que tienen estas herramientas se centra en el uso de un diccionario. El método habitual comienza con la búsqueda de una palabra en el mismo y, si esta es encontrada, se prosigue con la próxima; en caso contrario, se presenta al usuario una lista de palabras ordenadas de acuerdo a una determinada distancia entre la palabra analizada y las presentes en el diccionario, entre las cuales se espera esté la correcta. Por último, Padrón, Quesada, Pérez, González y Martínez (2014) señalan que el corrector ortográfico del procesador de textos encuentra errores tipográficos, porque compara cada palabra con su diccionario de referencia, pero no detecta errores que produce otra palabra bien escrita (por ejemplo, si se escribe *Pedro esta en el patio*).

Como es posible ver en las citas anteriores –de 1997, 2008 y 2014–, el funcionamiento de los correctores ortográficos no ha variado demasiado su esencia con los años, pero sí ha mejorado su funcionamiento en la detección de errores. Desgraciadamente, dichas herramientas son generalmente propietarias y privativas, por lo que no pueden adaptarse a proyectos científicos independientes como sería, por ejemplo –y centrándonos en nuestro caso–, el desarrollo de un sistema tutorial inteligente por un grupo de académicos de una universidad cualquiera. Además, para nuestra lengua –el español–, el trabajo de mejoramiento de estas herramientas no ha sido desarrollado al mismo nivel que para el inglés (Plüss & Pomponio, 2008). Por ello, surge la necesidad de aportar en dicho avance con el trabajo que se expone en las siguientes páginas.

2. El estudio: Diseño e implementación del corrector ortográfico

A continuación, se describe el diseño y la implementación del analizador y corrector ortográfico para el español, programado en *Python* 3, para el STI ELE-Tutora.

2.1. Construcción de un listado de lemas del español y su variabilidad léxica

Uno de los problemas que debe enfrentarse a la hora de trabajar en el desarrollo de un corrector ortográfico es la construcción de un lexicón que contenga todas las formas correctas de las palabras que existen en el español. Un listado de los lemas incluidos en el Diccionario de la Lengua Española de la Real Academia (DRAE) es insuficiente, ya que no se incluyen las formas derivadas (por ejemplo, aparece 'perro', pero no 'perrito' ni 'perritos'; aparece el verbo 'cantar', pero no sus conjugaciones en los diferentes modos y tiempos). Afortunadamente, proyectos como *FreeLing*

(http://nlp.cs.upc.edu/freeling/) o *Linguakit* (https://linguakit.com/es/) permiten la descarga de sus lexicones que, además, son *open source* (de código abierto). Dichos lexicones incluyen no sólo los lemas de las palabras del español sino que sus formas derivadas.

El lexicón utilizado en el presente trabajo se construyó en base a las dos fuentes ya mencionadas, además de un listado de todos los lemas del DRAE. La forma de combinar las tres fuentes señaladas en una sola fue a través de un *script* programado en *Python* 3. El funcionamiento de este requiere ubicar los textos que se desee utilizar como entrada en una carpeta llamada 'textos' (se pueden incluir todos los archivos de texto plano que se quiera) y ejecutar el *script*. Este devolverá una lista ordenada de las palabras únicas utilizadas, por lo que se incluirán una sola vez las palabras que se repitan en las fuentes empleadas.

Si bien el trabajo realizado por *FreeLing* y *Linguakit* es muy completo, es complejo afirmar que con esto se incluyan todas las formas posibles de las palabras en español. Por lo anterior, para completar aún más el listado resultante, se pueden añadir nuevas formas a este, mediante el procesamiento de un texto cualquiera (por ejemplo, obras literarias que estén en español y en un formato apropiado). Para ello el *script* construido es una herramienta fundamental. Así, por ejemplo, si se procesan diferentes textos y en estos aparece mil veces la palabra 'perritos', el *script* solo la incluirá una vez en el listado de salida, tal cual como se mencionó anteriormente.

Es necesario advertir que la acción de complementar el listado de formas del español, a través del procesamiento de distintos textos en el *script* construido, requiere mucha precaución. Hay que tener gran cuidado en que los textos de entrada estén correctamente escritos, ya que si alguno de ellos incluyera errores ortográficos, estos se traspasarían al listado de palabras correctas, lo que iría en desmedro del funcionamiento posterior del corrector ortográfico. En el caso específico del presente trabajo, como textos de entrada para complementar el listado de palabras correctas del español obtenido de las tres fuentes originales, se utilizaron varias obras en nuestra lengua que se pueden descargar desde el sitio web del Proyecto Gutenberg (http://www.gutenberg.org/browse/languages/es). Si bien hubo que revisar que los textos no estuvieran escritos en variantes antiguas del español, el proyecto ofrece una garantía –no absoluta, en todo caso– de que los textos están bien escritos y no se introducirán formas erróneas en el sistema.

Con todo el trabajo explicado hasta aquí, se logró un listado palabras correctas del español que contiene 591.526 términos. Como es de suponer, no se puede afirmar que este sea completo e incorpore todas las palabras de nuestra lengua y sus posibles derivaciones, más aún tratándose de nombres propios. Por ello, como se verá más adelante, el corrector ortográfico diseñado se planteó como un corrector dinámico, que permita agregar nuevas palabras al listado original.

2.2. Arquitectura y funcionamiento del corrector ortográfico

Primero, se presentará el diseño de la arquitectura propuesta para este (Figura 1), para posteriormente, abordar su construcción. Los números en el esquema se utilizan para apoyar la explicación de cada etapa que se dará en los puntos siguientes.

El trabajo se realizó en un equipo con Linux, pero el código final opera sin problemas en *Windows*. La entrada que utiliza el corrector ortográfico, empleando la terminología de *Python*, es una cadena de texto. Dicha cadena se procesa de dos formas, según se ve en la Figura 1: palabra a palabra (punto 1) y como una cadena en busca de patrones de error complejos (punto 8).

2.2.1. Etapa 1: Procesamiento palabra a palabra

En esta etapa, como se indica en el punto 2, la cadena de texto se divide en palabras independientes para su procesamiento. Esto se hace con la función *tokenize* incluida en NLTK (*Natural Language Toolkit*, disponible en http://www.nltk.org/), la que permite separar las palabras de una cadena de los signos de puntuación de la misma. Luego, se seleccionan solo los elementos que contienen grafemas del español, con sus correspondientes diacríticos si es que fuera el caso. Dicho de otra forma, se seleccionan solo los elementos que contengan las letras de nuestro alfabeto (incluida la eñe), más las vocales tildadas y la vocal ü (con diéresis).

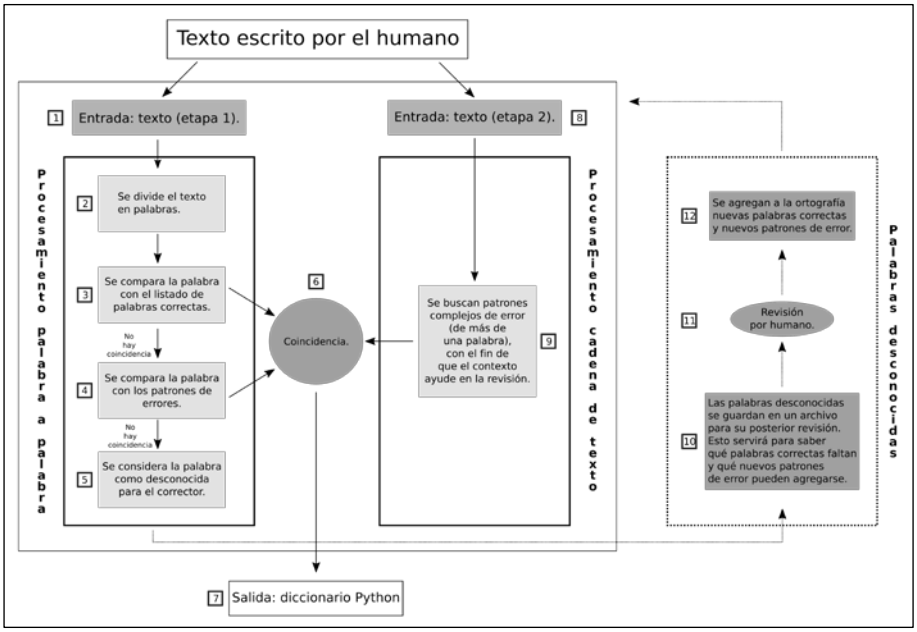

Figura 1. Arquitectura del corrector ortográfico.

Lo anterior deja fuera del análisis a los signos de puntuación, los números y cualquier otro signo ortográfico ajeno al español, como por ejemplo, los acentos breve (˘) y circunflejo (^). La idea de excluir los signos de puntuación, obviamente importantísimos en un corrector ortográfico, es porque esta primera etapa del procesamiento se centra solo en la escritura de cada palabra como unidad. De esta forma, por ejemplo, se busca detectar errores como escribir 'aviación' de la siguiente manera: 'aviasion'.

La revisión que realiza el corrector para cada palabra, comienza cotejando si esta se encuentra en el listado de palabras correctas del español (punto 3), que –como ya se dijo– contiene 591.526 términos. Siguiendo con el ejemplo anterior, si el término de entrada fuera 'aviación', el corrector lo encontraría en el listado aludido y lo reconocería como bien escrito. En este caso, al hallarse la coincidencia que se señala en el punto 6, la palabra se almacenaría como salida en un diccionario de *Python* llamado 'resultados simples', se etiquetaría en este diccionario como sin error (con la etiqueta 'SIN_ERROR') y se obviarían los pasos de los puntos 4 y 5 (más adelante se explicarán el funcionamiento del diccionario y las etiquetas en el corrector).

El listado de palabras correctas se construyó como un archivo de texto plano, en el cual las palabras están separadas por un salto de línea y tienen un espacio en blanco antes de cada una de ellas. El corrector, tomando este archivo como entrada de la información requerida, realiza una búsqueda para cotejar si la palabra se encuentra o no en el listado (el archivo en que se almacenan se llama 'listado_palabras_original'): si está en el listado la considera escrita correctamente, en caso contrario pasa a la etapa de búsqueda de errores, como se verá más adelante. El espacio en blanco y el salto de línea son una forma arbitraria de marcar el inicio y el final de cada palabra, con el fin de evitar que haya falsas coincidencias; si no existieran estos límites y, por ejemplo, se procesara la forma verbal mal escrita 'beo (veo)', el corrector arrojaría una coincidencia, entre otras, con las palabras 'beodo' o 'beorí' y consideraría a 'beo' como escrita correctamente.

Lo que se presentó anteriormente se basa en la siguiente función escrita en *Python* 3 (los números representan las líneas de código):

```
1  def palabras_correctas(patron):
2   with open(os.path.join(os.getcwd() +
    '/listado_palabras/listado_palabras_original'), 'r', encoding='utf8') as correctas:
3   for line in correctas:
4   if (' '+patron+'\n') in line:
5  return('encontrada')
```

En el caso de que no haya coincidencia en el cotejo del punto 3 (listado de palabras correctas), se pasa al punto 4. En este se compara la palabra con distintos patrones de error, construidos en base a expresiones regulares[3], que se almacenan en un archivo de

texto plano. Dichas expresiones se leen del archivo y se almacenan como tuplas[4] de *Python*, en un listado llamado 'errores_originales_simples'. La lectura de los patrones de error desde el archivo se realiza con el siguiente código:

```
1  errores_originales_simples = []
2  for linea in codecs.open(os.path.join(os.getcwd() +
   '/errores_originales/errores_simples'), 'r', encoding='utf8'):
3  tupla = ast.literal_eval(linea)
4  errores_originales_simples.append(tupla)
```

La función en que se basa el cotejo de las palabras de entrada con los patrones de error es la siguiente:

```
1  def corregir_ortografia_simples(sentencia):
2  outext = ''
3  error = ''
4  s = sentencia
5  for couple in buscar_reemplazar_simples:
6  correcta = re.sub(couple[0], couple[1], s)
7  incorrecta = re.search(couple[0], s)
8  if incorrecta:
9  return[incorrecta.group(0), correcta, couple[2]]
10 s=correcta
11 return ["", "", ""]
```

Un ejemplo abreviado de una expresión regular utilizada se presenta a continuación:

```
(r'"(^[Aa]bdica|^[Zz]onifica)+(cion$)"', r'"\1ción"', "ORTO_OMI_AGUDA"),
```

En el ejemplo anterior, se buscan las palabras 'abdicación' o 'zonificación' escritas sin tilde (con error). En caso de encontrar una coincidencia con alguna de ellas, se obtiene como salida una lista[5] de *Python* con el patrón encontrado (la palabra con error), la forma correcta y la etiqueta que identifica al error. Las etiquetas de error que se utilizaron son las propuestas por Ferreira et al. (2014), considerando únicamente las enfocadas en los errores ortográficos. Esta lista, al igual que las palabras correctas encontradas (como ya se explicó), se almacena en el diccionario de *Python* que arroja como salida final el corrector ortográfico. Un ejemplo de esta lista, para el caso de abdicación es:

```
['abdicacion', 'abdicación', 'OMI_ORTO_AGUDA']
```

En caso de no haber una coincidencia con algún patrón de error, y asumiendo que en el paso anterior tampoco la hubo con alguna palabra correcta, el sistema continúa a la siguiente etapa (punto 5). Aquí, los casos de palabras que no arrojan coincidencia en ninguno de los pasos anteriores, se etiquetan como palabras desconocidas para el

sistema (la etiqueta empleada es 'PALABRA_DESCONOCIDA'). Estas también se agregan al diccionario de *Python*, que contiene las salidas de las dos etapas anteriores.

El citado diccionario –junto con las listas y las tuplas– es una forma de almacenar datos en *Python* y se compone de dos elementos: clave y valor. Cada una de estas parejas debe tener siempre estos dos componentes y un diccionario puede incluir tantas parejas como se quiera, separadas unas de otras por una coma. A continuación se presenta un ejemplo de diccionario en *Python*:

diccionario = {"clave":"valor", "clave":"valor",...}

Los datos del diccionario[6] pueden ser recuperados posteriormente, con el fin de realizar las operaciones que se deseen, como por ejemplo, corregir el texto o entregar *feedback* a la persona que escribe sobre qué errores cometió. Con el fin de mostrar los resultados del corrector ortográfico, se ingresó en este el siguiente pasaje:

A la salida del estadio de Concepcion, los hinchas lanzaron piedras contra el bus de Wanderers.

Con el fin de tener una muestra de todas las posibilidades de salida en el diccionario resultante, se introdujo a propósito un error ortográfico al no tildar 'Concepción'. Además, el nombre 'Wanderers' es desconocido para el corrector pues no ha sido incorporado a este aún. La salida que se obtiene es la que sigue:

{'Concepcion': ['Concepcion', 'Concepción', 'OMI_ORTO_AGUDA'], 'de': 'SIN_ERROR', 'del': 'SIN_ERROR', 'hinchas': 'SIN_ERROR', 'la': 'SIN_ERROR', 'lanzaron': 'SIN_ERROR', 'bus': 'SIN_ERROR', 'contra': 'SIN_ERROR', 'salida': 'SIN_ERROR', 'los': 'SIN_ERROR', 'A': 'SIN_ERROR', 'Wanderers': 'PALABRA_DESCONOCIDA', 'el': 'SIN_ERROR', 'piedras': 'SIN_ERROR', 'estadio': 'SIN_ERROR'}

Como se puede ver en todos los casos, la clave es la palabra ingresada. El valor, por otro lado, en la mayoría de los casos es la etiqueta SIN_ERROR. En dos palabras, sin embargo, esto difiere. Como ya se adelantó, el nombre 'Wanderers' es desconocido para el corrector, por lo tanto, como valor se obtiene la etiqueta PALABRA_DESCONOCIDA. La otra salida distinta es justamente para la palabra en que se omitió tildar, esto es, 'Concepcion'. En este caso el valor es una lista de *Python* que contiene la palabra con el error, su forma correcta y la etiqueta que identifica el error, es decir, la omisión de la tilde en una palabra aguda: OMI_ORTO_AGUDA.

Todo lo explicado hasta aquí es para realizar el análisis de las palabras del texto aisladas, es decir, consideradas cada una de ellas como unidad.

2.2.2. Etapa 2: Procesamiento como una cadena en busca de patrones de error complejos

En el punto 8 de la Figura 1, se toma nuevamente el mismo texto de entrada, pero esta vez no se analiza palabra a palabra, sino que se revisa toda la cadena en busca de patrones de error complejos (punto 9) que necesitan de contexto para poder identificarlos. Por ejemplo, consideremos la siguiente oración como texto de entrada:

No se que pasa en México.

En el pasaje anterior se omitieron intencionadamente las tildes de 'se' y 'que'. Si esta oración se sometiera al análisis palabra a palabra, no arrojaría error alguno, ya que las formas 'se' y 'que' sin tilde existen en el español. Sin embargo, al leerlas en contexto podemos saber que 'se' no es un pronombre personal, sino que la primera persona del presente de indicativo del verbo saber; y 'que' es una forma interrogativa indirecta y no el pronombre relativo. Por lo tanto, si se busca 'no se que' como expresión regular, cuyo patrón complejo de error ha sido incorporado previamente al corrector, la aplicación puede reconocer que hay dos errores ortográficos en la oración y arrojar como salida la forma correcta 'no sé qué'. En el siguiente ejemplo se presenta la salida que arroja el corrector tras procesar el pasaje 'no se que pasa en México' (con los tildes omitidos en 'sé' y 'qué'). Como se puede ver, en primer lugar, la salida presenta 'no se que' con error y como una construcción compleja y después cada una de las palabras por separado son catalogadas como 'sin error', incluso a las que se omitió el tilde. Esto demuestra lo que permiten las reglas contextuales:

{'No se que': ['No se que', 'No sé qué', 'ORTO_OMI_TILDES'], 'pasa': 'SIN_ERROR', 'se': 'SIN_ERROR', 'en': 'SIN_ERROR', 'No': 'SIN_ERROR', 'que': 'SIN_ERROR', 'México': 'SIN_ERROR'}

Una de las limitaciones más obvias de estas reglas contextuales es la infinidad de combinaciones que ofrece el español, por lo que sería imposible preverlas todas; sin embargo, son una forma eficiente de solucionar casos como el recién expuesto y lograr que el corrector ortográfico funcione de forma más precisa. (Al igual que en el caso de las palabras aisladas, también pueden añadirse nuevos patrones complejos de error debido a la arquitectura dinámica del corrector, que se explicará en el apartado siguiente).En este caso, los patrones de error complejos también se almacenan en un archivo de texto y se leen con el siguiente código:

```
1 errores_originales_complejos = []
2 for linea in codecs.open(os.path.join(os.getcwd() +
  '/errores_originales/errores_complejos'), 'r', encoding='utf8'):
3 tupla = ast.literal_eval(linea)
4 errores_originales_complejos.append(tupla)
```

De lo expuesto en 2.2.1 y 2.2.2 se puede prever que el corrector adolecerá de dos problemas con total certeza: (1) no se puede asegurar que la lista de palabras correctas constituirá un listado total de toda la variabilidad léxica de los lemas del español, ni tampoco que contendrá los nombres propios que podría utilizar un hablante de nuestra lengua; (2) menos aún se puede afirmar que en los patrones simples de error (palabras) y complejos (contextuales) se contemplarán todas las posibilidades de error que podrían cometerse en nuestro idioma. Una dificultad adicional es que este corrector ortográfico está pensado para utilizarse en una aplicación montada en un servidor (específicamente, un sistema tutorial inteligente, aunque perfectamente podría adaptarse a otras aplicaciones por la versatilidad de *Python*), por lo que para modificarlo debería tenerse acceso directo al servidor. En caso de poder acceder, hay que tener en cuenta que muchos servidores están montados en *Linux* y muchas veces sin interfaz gráfica, por lo que cualquier operación se realiza a través de comandos de texto. Otra alternativa sería conectarse remotamente al servidor y editar el código del corrector, lo que tampoco es tarea sencilla.

Todo lo descrito en el párrafo anterior deviene en que el corrector, una vez terminado, se transformaría en una aplicación rígida, que sería muy difícil de modificar por quienes administren el sistema en que este se utilice. Por esto, se propone que el corrector tenga una estructura dinámica y que pueda ser modificado por una persona sin conocimientos de *Python* a través de una interfaz gráfica preparada para ello. Para lograrlo es imperativo contemplar esta posibilidad en la arquitectura del corrector y desarrollar el código necesario para este funcionamiento dinámico, y conectarlo posteriormente con la interfaz gráfica del área de administración que utilice la aplicación en que se inserte (a modo de ejemplo, podría utilizarse en un gestor de contenidos como *Moodle*, como es el caso de ELE-Tutora, o alguna aplicación hecha a medida).

2.2.3. Tratamiento de palabras desconocidas

Los puntos 10, 11 y 12 de la arquitectura propuesta (Figura 1) se enfocan en esta parte del corrector ortográfico. A partir del punto 5, cuando una palabra se califica como desconocida, puede ser por dos razones: la palabra está correctamente escrita, pero no se incluye en el listado de lemas y derivaciones que tiene el corrector; o la palabra contiene uno o más errores ortográficos, pero el patrón de error no está en el corrector. También puede darse el caso de que el sistema no arroje la etiqueta palabra desconocida, ya que para reconocer el error se necesita de un patrón complejo que no existe entre los incluidos (por ejemplo, que no hubiera una regla para reconocer 'no sé qué' cuando se omiten las tildes). Todos estos problemas pueden ser detectados por las personas que administren el sistema; sin embargo, solucionarlos no es algo sencillo, como ya se explicó.

El primer paso para enfrentar el punto descrito es que el *script* construido guarda las palabras etiquetadas como desconocidas en un archivo de texto plano, como un listado, separadas por saltos de línea (también podría ser separadas por comas, si se quisiera; se optó por el listado ya que facilita su lectura por un humano). De esta forma, este archivo puede ser fácilmente leído desde la interfaz gráfica antes planteada, para su revisión por los administradores del sistema *web* en que operará el corrector. A lo anterior, hay que agregar que el código que realiza la lectura de este archivo, de una forma análoga a la explicada para la creación del listado de palabras correctas, eliminará las palabras repetidas, para facilitar el trabajo del revisor. Desde esta misma interfaz -aclaramos, interfaz hipotética, pues en este trabajo solo se construyó el *script*-, quien administre podría seleccionar las palabras de ese listado que estén correctamente escritas y agregarlas a través de un formulario al archivo adicional de palabras correctas que posee el corrector ortográfico y, posteriormente, este las leerá para realizar el cotejo de las entradas de texto que reciba. La idea de crear un archivo adicional fue con el fin de mantener separado el listado original (denominado listado_palabras_original) del listado de palabras correctas agregadas posteriormente (denominado listado_palabras_agregadas), con el fin de mantener un mayor control sobre las modificaciones que se realicen (por ejemplo, que el revisor cometiera un error al agregar una palabra). Para lograr esto último, solo hubo que modificar la función palabras_correctas que se mostró anteriormente, para que quede de la siguiente forma:

```
1  def palabras_correctas(patron):
2      with open(os.path.join(os.getcwd() +
   '/listado_palabras/listado_palabras_original'), 'r', encoding='utf8') as
   correctas1, open(os.path.join(os.getcwd() +
   '/listado_palabras/listado_palabras_agregadas'), 'r', encoding='utf8') as
   correctas2:
3   for line in correctas1, correctas2:
4   if (' '+patron+'\n') in line:
5   return('encontrada')
```

Para el caso de añadir patrones simples de error, es decir a nivel de palabra, la forma elegida para realizarlo es agregando reglas por cada etiqueta de error, las que están previamente definidas con el fin de mantener el orden del sistema (como se señaló, se utilizaron las etiquetas propuestas por Ferreira et al. (2014). Estos patrones (expresiones regulares) agregados por los administradores o revisores del hipotético sistema también deben almacenarse en un archivo de texto plano; por lo mismo, se consideró un archivo por cada etiqueta (para mantener el orden, como ya se dijo), aunque perfectamente podrían manejarse todas las etiquetas en un único archivo. A continuación, se presenta el código para poder leer las expresiones regulares desde un archivo de texto, para la omisión de las tildes en palabras agudas, graves y esdrújulas:

```
1  errores_agregados_simples = []
2  for linea in codecs.open(os.path.join(os.getcwd() +
   '/errores_nuevos/patrones_nuevos_ORTO_OMI_AGUDA'), 'r',
   encoding='utf8'):
3  tupla = ast.literal_eval(linea)
4  errores_agregados_simples.append(tupla)
5  for linea in codecs.open(os.path.join(os.getcwd() +
   '/errores_nuevos/patrones_nuevos_OMI_ORTO_LLANA'), 'r',
   encoding='utf8'):
6  tupla = ast.literal_eval(linea)
7  errores_agregados_simples.append(tupla)
8  for linea in codecs.open(os.path.join(os.getcwd() +
   '/errores_nuevos/patrones_nuevos_ORTO_OMI_ESDRUJULA'), 'r',
   encoding='utf8'):
9  tupla = ast.literal_eval(linea)
10 errores_agregados_simples.append(tupla)
```

El citado código lee los patrones de error almacenados en los siguientes archivos:

- patrones_nuevos_ORTO_OMI_AGUDA
- patrones_nuevos_ORTO_OMI_LLANA
- patrones_nuevos_ORTO_OMI_ESDRUJULA

Para el caso de los patrones complejos de error que se añadan se optó por almacenarlos en un único archivo de texto plano llamado patrones_nuevos_COMPLEJOS. El código utilizado para leerlo es:

```
1  errores_agregados_complejos = []
2  for linea in codecs.open(os.path.join(os.getcwd() +
   '/errores_nuevos/patrones_nuevos_COMPLEJOS'), 'r', encoding='utf8'):
3  tupla = ast.literal_eval(linea)
4  errores_agregados_complejos.append(tupla)
```

2.3. Entrega de resultados

Como resumen de lo hasta aquí presentado, el corrector ortográfico en primer lugar coteja las palabras de entrada con un listado de palabras correctas. Luego busca los patrones de error almacenados en las siguientes variables creadas en *Python*:

- errores_originales_simples = []
- errores_originales_complejos = []
- errores_agregados_simples = []
- errores_agregados_complejos = []

Sin embargo, y para respetar el orden de procesamiento, previo al trabajo recién descrito, el corrector combina, por una parte, los patrones de error simples (originales y agregados) en la variable 'buscar_reemplazar_simples'; y, por otra, los patrones de error complejos (originales y agregados), en la variable

'buscar_reemplazar_complejos'. Lo anterior, con el fin de facilitar el procesamiento posterior. El código que efectúa la tarea descrita es el que sigue:

```
1 buscar_reemplazar_simples =
  errores_originales_simples+errores_agregados_simples
2 buscar_reemplazar_complejos =
  errores_originales_complejos+errores_agregados_complejos
```

Por último, en base a estas dos variables, el corrector realiza las tareas ya descritas a lo largo de los puntos precedentes. Para ello, primero ejecuta el procesamiento palabra a palabra que busca si cada término de la entrada está presente en los listados de palabras correctas (originales y agregadas). Luego pasa a revisar si están presentes los patrones simples de error (originales y agregados). Finalmente, en caso de que no haya resultados en las dos etapas previas, asume que la palabra es desconocida para el sistema. Los resultados de todo lo anterior, los guarda en un diccionario llamado 'resultados_simples'. El código que se presenta a continuación realiza las tareas descritas:

```
1  resultados_simples = {}
2  with open('entrada', 'r') as filer:
3  tokenizar = nltk.word_tokenize(filer.read())
4  regex = re.compile('(^[a-zñA-ZÑáéíóúüÁÉÍÓÚÜ]+$)')
5  limpiar = [m.group(1) for l in tokenizar for m in [regex.search(l)] if m]
6  texto_limpio = "\n".join(limpiar)
7  for palabra in texto_limpio.split():
8  if palabras_correctas(palabra) == 'encontrada':
9  resultados_simples.update({palabra: 'SIN_ERROR'})
10 elif (corregir_ortografia_simples(palabra))[0] == palabra:
11 resultados_simples.update({palabra: (corregir_ortografia_simples(palabra))})
12 else:
13 resultados_simples.update({palabra: 'PALABRA_DESCONOCIDA'})
14 with open(os.path.join(os.getcwd() + '/listado_palabras/desconocidas'), 'a') as
   descon:
15 descon.write(palabra+'\n')
```

En el caso de los patrones complejos de error, el siguiente código ejecuta la tarea de buscarlos y almacena sus resultados en el diccionario 'resultados_complejos':

```
1  resultados_complejos = {}
2  with open('entrada', 'r') as filer:
3  tokenizar = nltk.word_tokenize(filer.read())
4  regex = re.compile('(^[a-zñA-ZÑáéíóúüÁÉÍÓÚÜ]+$)')
5  limpiar = [m.group(1) for l in tokenizar for m in [regex.search(l)] if m]
6  texto_limpio_cadena = " ".join(limpiar)
7  for couple in buscar_reemplazar_complejos:
8  correcta = re.sub(couple[0], couple[1], texto_limpio_cadena)
9  incorrecta = re.search(couple[0], texto_limpio_cadena)
```

```
10 if incorrecta:
11 salida = [incorrecta.group(0), correcta, couple[2]]
12 resultados_complejos.update({incorrecta.group(0): salida})
```

Finalmente, se combinan los resultados de los dos diccionarios que se obtienen de los pasos anteriores, en un único diccionario llamado simplemente 'resultados'. Este es la salida final de todo el procesamiento realizado. El código para este último paso es:

```
resultados = dict(resultados_simples, **resultados_complejos)
```

Como salida arroja una similar a la ya presentada en el ejemplo expuesto en 2.2.1 que se reproduce nuevamente:

Entrada:

A la salida del estadio de Concepcion, los hinchas lanzaron piedras contra el bus de Wanderers.

Salida:

{'Concepcion': ['Concepcion', 'Concepción', 'OMI_ORTO_AGUDA'], 'de': 'SIN_ERROR', 'del': 'SIN_ERROR', 'hinchas': 'SIN_ERROR', 'la': 'SIN_ERROR', 'lanzaron': 'SIN_ERROR', 'bus': 'SIN_ERROR', 'contra': 'SIN_ERROR', 'salida': 'SIN_ERROR', 'los': 'SIN_ERROR', 'A': 'SIN_ERROR', 'Wanderers': 'PALABRA_DESCONOCIDA', 'el': 'SIN_ERROR', 'piedras': 'SIN_ERROR', 'estadio': 'SIN_ERROR'}

A este diccionario de *Python* que constituye la salida se aplican los procedimientos descritos en 2.3 que hacen que este corrector ortográfico sea dinámico, y pueda mejorar su funcionamiento con el tiempo y la ayuda de un revisor humano. Por último, es importante señalar que el sistema comenzó con 860 patrones de error simples, a los que se han añadido posteriormente otros 173. En el caso de los patrones de error complejos, hay 52 que son originales y 86 que se han agregado posteriormente. En el caso de los patrones de error simples hay muchos que sirven para detectar errores en más de una palabra como en el ejemplo que se presentó anteriormente:

```
(r'"(^[Aa]bdica|^[Zz]onifica)+(cion$)"', r'"\1ción"', "ORTO_OMI_AGUDA"),
```

En este, por motivos de espacio, se suprimieron todas las palabras que están entre 'abdicación' y 'zonificación', ya que el patrón completo revisa la ausencia de tilde en 'ción' para 2.427 palabras.

2.4. Funcionamiento del corrector ortográfico en ELE-Tutora

El corrector ortográfico construido está integrado en el STI ELE-Tutora, que es un sistema destinado a la enseñanza del español como lengua extranjera. Si bien el corrector ortográfico opera sobre todas las entradas de texto que el estudiante realice en el STI y les entrega retroalimentación sobre sus errores cuando la planificación de

ELE-Tutora así lo requiere, su importancia es aún mayor en el funcionamiento del tutor ortográfico que incorpora el STI. En la Figura 2 se puede ver la portada de este, en la sección enfocada en la ortografía acentual:

Figura 2. Portada del Sistema Tutorial Ortográfico de ELE-Tutora.

En la Figura 3 se presenta un esquema del funcionamiento del corrector ortográfico integrado en el STI ELE-Tutora.

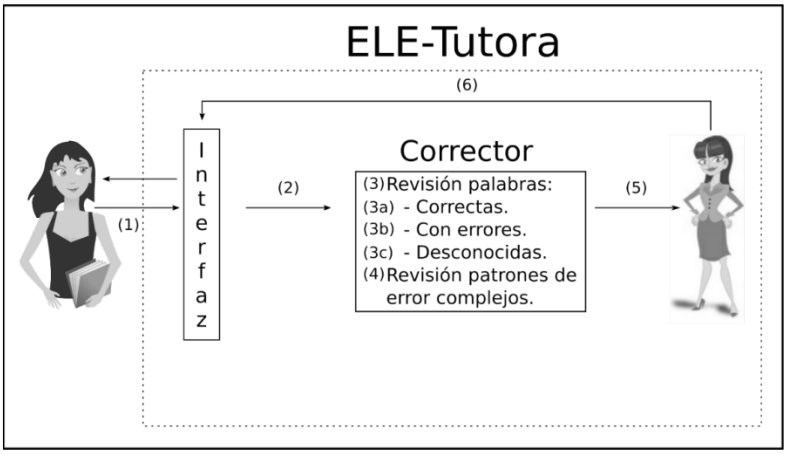

Figura 3. Esquema de funcionamiento del corrector ortográfico en ELE-Tutora.

El funcionamiento del corrector en ELE-Tutora, que se presenta en la figura precedente, puede explicarse de la siguiente forma: (1) el estudiante ingresa una cadena de texto a través de la interfaz y (2) esta cadena constituye la entrada del corrector ortográfico. Este analiza la cadena (3) palabra a palabra buscando si (3a) están escritas correctamente, (3b) si presentan errores o (3c) si son desconocidas para el corrector, para luego analizar la cadena completa (4) en busca de patrones de error complejos. Finalmente, (5) arroja la salida en forma de un diccionario de *Python*, que permite a

ELE-Tutora elaborar la (6) retroalimentación que se le entregará al estudiante. Por último, en la Figura 4 se presenta una imagen de un ejercicio del tutor ortográfico de ELE-Tutora.

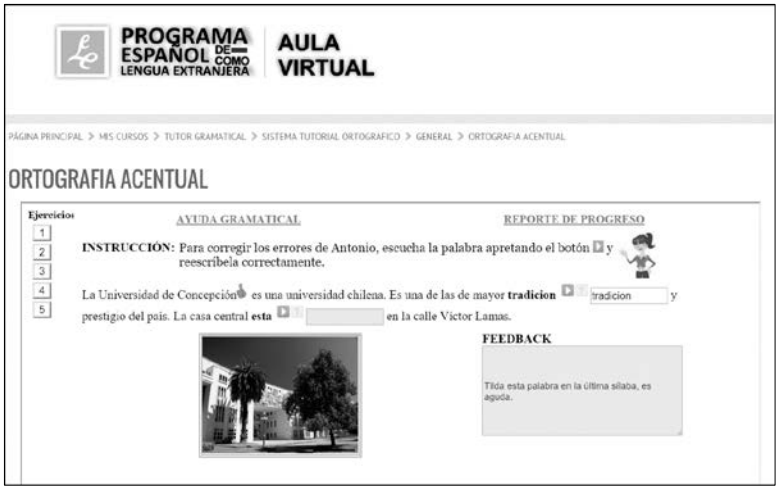

Figura 4. Ejercicio del tutor ortográfico de ELE-Tutora.

El ejercicio presentado de ortografía acentual le entrega al estudiante un pasaje de texto, en que se destacan tres palabras a las que se le suprimió la tilde. La tarea del alumno consiste en escuchar un archivo de audio que le ofrece ELE-Tutora y, posteriormente, escribir correctamente la palabra en el espacio asignado para ello. En el ejemplo concreto de la Figura 4, el estudiante ya respondió correctamente la primera palabra –Concepción– y cometió un error al escribir sin tilde la segunda palabra –tradición–. Ante esto, ELE-Tutora le ofrece *feedback* al alumno y le recomienda que 'tilde esta palabra en la última sílaba, es aguda'.

Lo anterior demuestra la integración del corrector ortográfico en el STI, ya que gracias a la salida que este entrega en forma de diccionario de *Python*, es posible que otro componente de ELE-Tutora genere el *feedback* que se le entrega al alumno con el fin de que este pueda reparar el error cometido.

CONCLUSIONES

En relación con el planteamiento del corrector ortográfico realizado en los puntos precedentes y la construcción del mismo que se llevó a cabo, es necesario precisar sus limitaciones, fundamentalmente en lo que se refiere a su funcionamiento. El corrector ortográfico no es capaz de reconocer todos los errores de esta índole posibles de cometer en la lengua española. De hecho, es probable que tal corrector ortográfico no haya sido construido aún, ya que debería adaptarse a las infinitas alternativas de error que puede cometer una persona al escribir un texto en nuestra lengua. No obstante, lo anterior, no quiere decir que se estime que la herramienta diseñada para revisar la

ortografía de los textos sea pobre; por el contrario, se reconoce su potencia, sobre todo, por el hecho de ser un corrector dinámico, que puede ser adaptado a los escenarios que vayan surgiendo en la aplicación concreta que se haga de la herramienta (en nuestro caso, formar parte de un sistema tutorial inteligente).

En este sentido, se considera al corrector construido una herramienta que puede prestar ayuda en el mejoramiento de la precisión lingüística en la producción escrita de un texto, por dos razones principales: una de corte técnico y otra del ámbito lingüístico. La primera de estas es que su construcción en base a herramientas consideradas como *software* libre –*Python* y NLTK– permiten utilizarlo en cualquier proyecto, adaptándolo con facilidad y sin la limitación que imponen las licencias comerciales. La segunda razón es que su arquitectura dinámica permite mejorar la precisión de su funcionamiento, volviéndolo más eficiente en la detección de errores en el ambiente en que se inserte, por ejemplo, un STI como ELE-Tutora. En este punto, hay que tener claridad que el corrector requiere de módulos adicionales –que existen en un STI, por ejemplo– que interpreten la salida que entregan los diccionarios de *Python* y la transformen en retroalimentación que pueda ser útil al usuario del sistema, permitiéndole de esta forma mejorar el texto escrito.

Acerca de las mejoras que se pretenden realizar a futuro al corrector, una de las ideas que se maneja, es incorporar en su funcionamiento un analizador sintáctico o *parser*, con el fin de poder determinar la categoría gramatical de las diferentes palabras, para desambiguar de una manera más precisa y funcional cadenas confusas como la citada 'no sé qué' (omitiendo uno o los dos tildes por error). Lo recién descrito sería un paso importante hacia potenciar aún más el funcionamiento de la herramienta, con el fin de volverla más precisa y ampliar su espectro de reconocimiento de errores. Pero, como se dijo, es algo en lo que aún se debe trabajar en su planificación y, posteriormente, en su desarrollo e implementación. La idea es dejar constancia de que el corrector construido se considera una herramienta potente, pero perfectible y es una tarea que queda abierta y planteada.

REFERENCIAS BIBLIOGRÁFICAS

Alexopoulou, A. (2005). El error: Un concepto clave en los estudios de adquisición de segundas lenguas. *Revista de Lingüística Teórica y Aplicada, 43*(1), 75-92.

Consejo de Europa (2002). *Marco común europeo de referencia para las lenguas*. Madrid: Anaya.

De Alba Quiñones, V. (2009). El análisis de errores en el campo de ELE. Algunas cuestiones metodológicas. *Revista Nebrija de Lingüística aplicada a la enseñanza de Lenguas*, *5*(3), 1-16.

Ellis, R. (1997). *Second language acquisition*. Oxford: Oxford University Press.

Ferreira, A. (2007). Estrategias efectivas de *feedback* correctivo para el aprendizaje de lenguas asistido por computadores. *Revista Signos. Estudios de Lingüística*, *40*(65), 521-544 [en línea]. Disponible en: http://www.scielo.cl/scielo.php?pid=S0718-09342007000300007 &script=sci_arttext

Ferreira, A. (2014a). *El feedback correctivo escrito directo e indirecto en la adquisición y aprendizaje del español como lengua extranjera.* Proyecto Fondecyt 1140651, Chile: Conicyt.

Ferreira, A. (2014b). *ELE-TUTORA: Un Sistema Tutorial Inteligente para el Español como Lengua Extranjera.* Concepción, Chile: Universidad de Concepción.

Ferreira, A. (2015). *Corpus CAELE: Un Corpus de Aprendientes de Español como Lengua Extranjera en formato electrónico*. Concepción, Chile: Universidad de Concepción.

Ferreira, A. & Atkinson, J. (2009). Designing a feedback component of an intelligent tutoring system for foreign language. *Knowledge-Based Systems Journal, Elsevier*, *22*(7), 496-501.

Ferreira, A. & Kotz, G. (2010). ELE-Tutor Inteligente: Un analizador computacional para el tratamiento de errores gramaticales en Español como Lengua Extranjera. *Revista Signos. Estudios de Lingüística, 43*(73), 211-236.

Ferreira, A., Moore, J. & Mellish, Ch. (2007). A study of feedback strategies in foreign language classrooms and tutorials with implications for Intelligent Computer-Assisted Language Learning Systems. *International Journal of Artificial Intelligence in Education, 17*(4), 389-422.

Ferreira, A., Elejalde, J. & Vine, A. (2014). Análisis de Errores Asistido por Computador basado en un Corpus de Aprendientes de Español como Lengua Extranjera. *Revista Signos. Estudios de Lingüística, 47*(86), 385-411.

Ferreira, A., Salcedo, P., Kotz, G. & Barrientos, F. (2012). La arquitectura de ELE-TUTOR: Un Sistema Tutorial Inteligente para el Español como Lengua Extranjera. *Revista Signos. Estudios de Lingüística, 45*(79), 102-131.

Padrón, C. I., Quesada, N., Pérez, A., González, P. L. & Martínez, L. E. (2014). Aspectos importantes de la redacción científica. *Revista de Ciencias Médicas de Pinar del Río*, *18*(2), 362-380.

Plüss, B. & Pomponio, L. (2008). *Tratamiento automático de reglas ortográficas para la detección y corrección de errores*. 37th Argentine Conference on Informatics and Operations Research [en línea]. Disponible en: http://www.open.ac.uk/blogs/brianpluss/wp-content/uploads/2011/07/corrector-EST-short.pdf

San Mateo, A. (2016). Un corpus de bigramas utilizado como corrector ortográfico y gramatical destinado a hablantes nativos de español. *Revista Signos. Estudios de Lingüística, 49*(90), 94-118.

Tapia, A. & Ariza, A. (1997-1998). El corrector ortográfico y la presentación del texto escrito. *Cauce: Revista de filología y su didáctica, 2*(20-21), 375-412. ISSN 0212-0410.

Vázquez, G. (2009). Análisis de errores, el concepto de corrección y el desarrollo de la autonomía. *Revista Nebrija de Lingüística Aplicada a la Enseñanza de Lenguas*, 5, 113-122.

Venkatesh, R., Naganathan, E. R. & Uma Maheswari, N. (2010). Intelligent Tutoring System Using Hybrid Expert System With Speech Model in Neural Network. *International Journal of Computer Theory and Engineering, 2*(1) [en línea]. Disponible en: http://www.ijcte.org/papers/108-G225.pdf

NOTAS

[1] ELE-TUTORA es el mismo Sistema Tutorial Inteligente que, anteriormente, fue conocido como ELE-TUTOR.

[2] Fondecyt regular, 1110812 (*Un Sistema Tutorial Inteligente para la Focalización en la Forma en la Enseñanza del Español como Lengua Extranjera*, 2011 a 2014) y Fondecyt regular, 1140651 (*El feedback correctivo escrito directo e indirecto en la adquisición y aprendizaje del español como lengua extranjera*, desde marzo de 2014).

[3] Las expresiones regulares (llamadas RE –por sus iniciales en inglés– o patrones de expresiones regulares) son, esencialmente, un sencillo y altamente especializado lenguaje de programación incrustado dentro de *Python* y puesto a disposición a través del módulo re. Al usar este sencillo lenguaje, se deben especificar las reglas para el conjunto de posibles cadenas que se desean buscar; este conjunto puede contener frases en inglés (o en otra lengua), o direcciones de correo electrónico, o lo que se quiera. También se pueden utilizar las expresiones regulares para modificar una cadena de texto o dividirla de diferentes maneras (documentación de *Python Software Foundation*, disponible en: https://docs.python.org/3/howto/regex.html?highlight=regular%20expression).

[4] Las tuplas son secuencias inmutables, normalmente utilizadas para almacenar colecciones de datos heterogéneos y se escriben entre paréntesis normales: (). Las tuplas también se utilizan para casos donde se requiere una secuencia inmutable de datos homogéneos (documentación de *Python Software Foundation*, disponible en: https://docs.python.org/3/library/stdtypes.html?highlight=tuple#tuple).

[5] Las listas de *Python* son un tipo de datos compuestos –el más versátil de los que incluye este lenguaje–, que se utilizan para agrupar otros valores. Se escriben como una lista de valores separados por comas (ítems) encerrados entre corchetes: []. Las listas pueden contener elementos de diferentes tipos pero, por lo general, se utilizan para agrupar elementos de un mismo tipo. Por ejemplo, un listado de números de teléfonos móviles: [98563487, 96846389, 99140711] (documentación de *Python Software Foundation*, disponible en https://docs.python.org/3.5/tutorial/introduction.html#lists).

[6] Para ampliar un poco más la explicación dada, se puede decir que la mejor forma de concebir un diccionario de *Python*, es pensar en este como un conjunto desordenado de claves y valores, con el requisito de que las claves sean únicas (dentro de un diccionario). Un par de llaves crea un diccionario vacío: {}. Dentro de estos, se colocan separados por dos puntos las claves y sus correspondientes valores y cada una de estas parejas se separa de otras mediante comas. Las principales operaciones en un diccionario son almacenar un valor con alguna clave y extraer el valor utilizando la clave. Los diccionarios pueden incluir listas, tuplas o, incluso, otros diccionarios. Por ejemplo, imaginemos utilizar un diccionario de *Python* para almacenar números de teléfonos, asociados a los nombres de sus propietarios; así cada nombre propio será una clave y el número de teléfono su valor: {'pedro': 98563487, 'juan': 96846389, 'diego': 99140711} (documentación de *Python Software Foundation*, disponible en: https://docs.python.org/3.5/tutorial/datastructures.html#dictionaries).

* AGRADECIMIENTOS

El presente artículo se desarrolló en el contexto del proyecto de investigación FONDECYT 1140651: "El *feedback* correctivo escrito directo e indirecto en la adquisición y aprendizaje del español como lengua extranjera". (Investigadora Responsable: Dra. Anita Ferreira Cabrera).