Escuela Técnica Superior de Ingenieros Informáticos

Universidad Polítecnica de Madrid

TweetSC: Corrector de texto para Twitter

Trabajo Fin de Máster

Máster Universitario en Inteligencia Artificial

AUTOR: Javier Moreno Vega
TUTOR/ES: Óscar Corcho García y
Víctor Rodríguez Doncel

https://jmorenov.github.io/TweetSC/

6 de julio de 2018

RESUMEN

Esta memoria describe TweetSC, un corrector de texto para mensajes en español en Twitter. Debido a que los nuevos sistemas de comunicación han generado un uso diferente del idioma, ha surgido un problema en el análisis de textos. Entre todas las redes sociales este trabajo se centra en Twitter debido a las características que tiene. El objetivo principal de este trabajo es la creación de un corrector para tweets en español. El estado del arte sobre este tema no es muy amplío, y en español aún menos; los enfoques que se suelen seguir son los de normalización y adaptación de herramientas, aunque en los últimos años ha surgido un nuevo enfoque basado en redes neuronales y vectores de palabras. Mi solución, a la que he llamado "Tweet Spell Checker" (TweetSC),consiste en un sistema basado en modulos que se ejecutan de forma secuencial, desde que entra el texto sin normalizar se va aplicando módulo a módulo hasta que se devuelve el texto normalizado. Los módulos construidos son: Tokenizador, Reglas de preproceso, Detector de palabras OOV (Out-Of-Vocabulary), generador de candidatos para cada OOV, ranking de candidatos y postproceso; además cada modulo implementa de forma interna varios métodos que se pueden quitar o añadir. La implementación se ha desarrollado en Java y se ha dividido en tres paquetes: tweetsccore (núcleo del sistema y que funciona como una biblioteca por sí sola), tweetscexecutable (paquete que genera un ejecutable jar para su uso desde línea de comandos) y tweetscweb (aplicación web [1]). Los resultados se han comparado con los de Tweet-Nor 2013 [3]. He conseguido unos resultados de accurancy en general bajos pero con posibilidades de mejora y líneas futuras.

[1] https://jmorenov.github.io/TweetSC/

SUMMARY

This dissertation describes TweetSC, a text corrector for Spanish messages on Twitter. Because the new communication systems have generated a different use of the language, a problem has arisen in the analysis of texts. Among all social networks, this work focuses on Twitter due to the characteristics it has. The main objective of this work is the creation of a corrector for tweets in Spanish. The state of the art on this subject is not very extensive, and in Spanish even less; the approaches that are usually followed are those of standardization and adaptation of tools, although in recent years a new approach based on neural networks and word vectors has emerged. My solution, which I have called " Tweet Spell Checker " (TweetSC), consists of a system based on modules that are executed sequentially, since the text enters without normalizing it is applied module to module until it is returned the normalized text. The built modules are: Tokenizer, Preprocessing rules, OOV (Out-Of-Vocabulary) word detector, generator of candidates for each OOV, ranking of candidates and post-processing; In addition, each module internally implements several methods that can be removed or added. The implementation has been developed in Java and has been divided into three packages: tweetsccore (core of the system and that works as a library by itself), tweetscexecutable (package that generates an executable jar for use from the command line) and tweetscweb (web Application [2])). The results have been compared with those of Tweet-Nor 2013 [3]. I have achieved generally low accurancy results but with possibilities for improvement and future lines.

[2] https://jmorenov.github.io/TweetSC/

Índice

Índice de figuras

Índice de cuadros

1. Introducción

1.1. Motivación

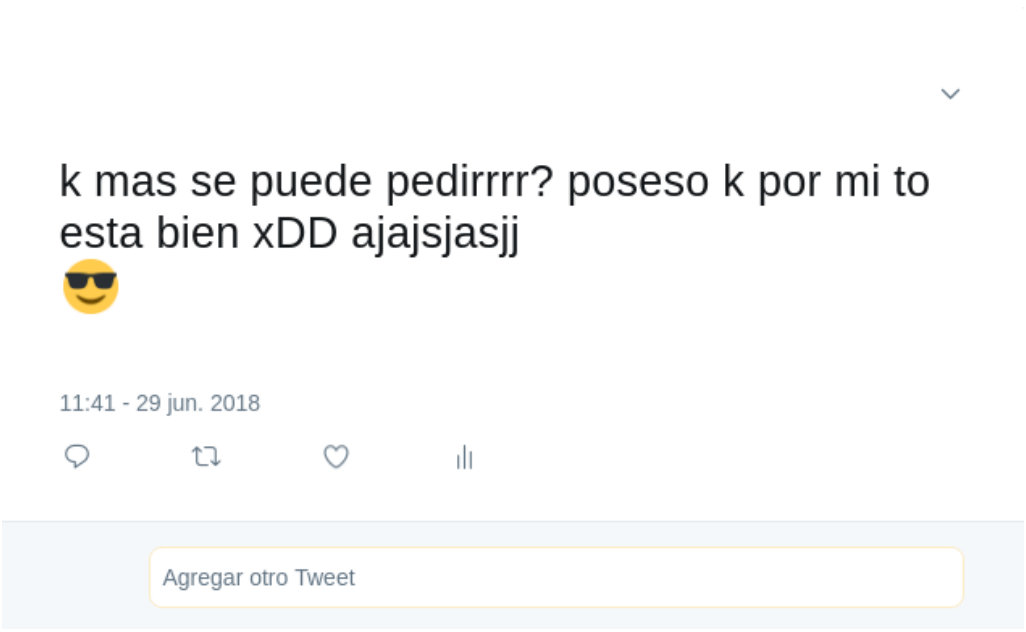

Fig. 1: EJemplo de tweet mal escrito.

Los nuevos sistemas de comunicación como la mensajería instantánea, los chats, las redes sociales han generado un uso diferente de los idiomas en estos ámbitos, llamado lenguaje tipo chat [46]. Una de estas redes sociales y en la que este trabajo va a centrarse es Twitter. En esta red social predomina el uso de emoticonos, repetición de vocales o eliminación de las mismas, uso abusivo de mayúsculas o su asusencia, siglas de expresiones populares; y otras características que dificulta el análisis de los textos. Las ventajas que ofrece esta red social para investigar sobre ella son la existencia de cantidad de datos en tiempo real y su fácil acceso.

Uno de los principales problemas a la hora de analizar textos procedentes de las redes sociales son los errores ortográficos y gramaticales que suelen contener, así como la presencia de elementos propios de este tipo de foros que requieren de un procesamiento especial (i.e. *hashtags*, formas de mencionar a otros usuarios o emoticonos y expresiones habituales en las redes). Además, la limitación en el número de caracteres existente en Twitter la convierte en un caso singular dentro de las redes sociales, ya que los usuarios tienden a adaptar su forma de escribir a dicha limitación, omitiendo palabras y creando abreviaturas que dificultan el uso de herramientas genéricas de procesamiento del lenguaje, especialmente a la hora de realizar tareas como el Análisis de Sentimientos.

Los usuarios en Twitter tienden a cometer errores tipográficos, utilizar abreviaturas (tfno.), abreviaciones(telefon), realizar sustituciones fonéticas y emplear estructuras no gramaticales en los mensajes cortos de texto, dificultando la tarea de las herramientas de análisis.

Esto es lo que se consideran palabras mal formadas. La detección de las palabras mal formadas es difícil debido al contexto ruidoso y limitado de Twitter, y su normalización es el objetivo de este trabajo.

La tarea de normalización también es beneficiosa para un estudio de marcas o personas y sobre lo que las persones opinan sobre ello en las redes social, ya que sin el proceso de normalización y análisis de sentimientos estaríamos ante millones de datos que costarían mucho trabajo analizar de una forma automática.

1.2. Objetivos

El objetivo principal de este trabajo es la creación de un corrector que "normalice" tweets en español.
Para cumplir con este objetivo principal se ha dividido en los siguientes subobjetivos.

- Acceso a la API de Twitter para obtener tweets.
- Tokenización de tweets.
- Detección entre los tokens las palabras fuera del vocabulario (*Out-of-Vocabulary*, OOV).
- Anotar el tipo de palabras OOV.
- Corrección de palabras OOV.

Estos subobjetivos se han cumplido con su implementación en un módulo software que además estará disponible en una aplicación web [41].

También he ejecutado este corrector sobre un corpus de tweets [3] y se han comparado los resultados con los que se consiguieron en Tweet-Norm 2013 [3].

1.3. Resumen del documento

Esta memoria explica todo el trabajo desarrollado entrando en detalle en el estado del arte y el módulo software desarrollado. Primero se realiza una introducción al tema y se exponen los objetivos a realizar. En segundo lugar se presenta el estado del arte sobre el tema de la corrección de textos, específicamente en Twitter y en español. Posteriormente explicamos la solución propuesta con todas sus fases. A continuación se presenta la implementación desarrollada y la documentación del código,

[3] http://komunitatea.elhuyar.eus/tweet-norm/

además de los recursos utilizados. En siguiente lugar evaluamos los resultados. Y por último se desarrollan las conclusiones y líneas futuras.

2. Estado del arte

2.1. Introducción

En la actualidad, la normalización lingüística de tweets [28] supone un campo de gran interés y en donde la mayoría de trabajos se han realizado sobre textos en inglés y pocos en español. Además no hay ningún trabajo en donde se incluya, dentro de la normalización de tweets, el estudio de los *hashtag* o etiquetas y los emoticonos, y su contexto.
Una introducción al tema de normalización de tweets es el artículo de Eisenstein [15], donde se revisa el estado del arte en NLP sobre variantes SMS y tweets, y cómo la comunidad científica ha respondido por dos caminos: normalización y adaptación de herramientas.

2.2. Normalización

El modelo de canal ruidoso [59] ha sido tradicionalmente la primera aproximación a la normalización de textos. Supone que el texto mal formado es T y su forma normalizada es S, por lo que hay que encontrar: arg max $P(S|T)$, calculando arg max $P(T|S)P(S)$, P(S) es el modelo del lenguaje y $P(T|S)$ es el modelo de error. Brill y Moore [8] caracterizan el modelo de error calculando el producto de operaciones de probabilidad en partes de cadenas de caracteres. Toutanova y Moore [63] mejoraron el modelo incorporando información de la pronunciación. Choudhury et al. [10] modelan el proceso de generación de texto a nivel de palabra para mensajes SMS considerando las abreviaturas grafémicas/fonéticas y los errores tipográficos involuntarios como transiciones de estado ocultas del modelo de Markov (HMM) y emisiones, respectivamente. Cook y Stevenson [11] expandieron el modelo de error introduciendo inferencias de diferentes procesos de formación erróneos, de acuerdo con la distribución de errores muestreada.

Mientras el modelo de canal ruidoso es apropiado para normalización de textos, es difícil aproximar la normalización con exactitud, además estos métodos ignoran el contexto alrededor del OOV, el cual ayuda a resolver ambigüedades. La traducción automática estadística (SMT) se ha propuesto como un medio de normalización de texto sensible al contexto, al tratar el texto mal formado como el idioma de origen, y la forma estándar como el idioma de destino. Por ejemplo Aw et al. [5]. Kobus et al. consideraron la normalización de textos como un problema de reconocimiento de voz [34]. Beaufort et al. [6] métodos de estado finitos combinando las ventajas de SMS y el modelo de canal ruidoso. Kaufmann y Kalita [33] usan un enfoque de traducción automática con un preprocesador para la normalización sintáctica (en lugar de léxica).

El problema de estos trabajos anteriores es que requieren datos de entrenamiento anotados a gran escala, lo que limita su adaptabilidad a nuevos dominios o idiomas, mientras que los trabajos: Whitelaw et al. [67] y Han et al.[28], no. Estos trabajos

son una buena referencia en el campo de la normalización de tweets en inglés de forma no supervisada. En donde para detectar palabras fuera de diccionario (OOV) utilizan la GNU aspell [4], y los usuarios (@usuario), los *hashtags* y las URLs son excluidas de la normalización. La normalización tiene relación con los correctores de texto [50] pero difiere en que las palabras mal formadas en los mensajes de texto suelen ser intencionadas, para ahorrar caracteres, como identidad social, o debido a la convención en este subgénero de texto. La detección de las palabras mal formadas es difícil debido al contexto ruidoso. El objetivo es normalizar estas palabra mal formadas, además muchas palabras mal formadas son ambiguas y requieren el contexto para poder normalizarlas.

2.3. Adaptación de herramientas

En vez de adaptar el texto a herramientas de análisis otro de los caminos a seguir es adaptar las herramientas de análisis al texto. Destacan los trabajos de reconocimiento de voz de Gimpel et al. [24] y Owoputi et al. [49]; reconocimiento de entidades: Finin et al. [17], Ritter et al. [53], Liu et al. [36]; análisis gramatical: Foster et al. [20]; modelización de diálogos: [52] y resumen automático de textos de Sharifi et al. [60].

El reconocimiento de entidades nombradas (NER) es una tarea de extracción de información que busca localizar y clasificar en categorías predefinidas, como personas, organizaciones, lugares, expresiones de tiempo y cantidades, entidades encontradas en un texto. Las soluciones propuestas para NER suelen recaer en tres categorías: *Basado en reglas* [35], *Basada en aprendizaje automático* [19] [61] y *Métodos híbridos* [32]. Con la disponibilidad de datos anotados, Enron [39] y CoNLL03 [57] se han convertidos en los nuevos métodos dominantes. El estudio actual NER se centra principalmente en textos formales, de hecho, el estado del arte actual (CoNLL03) tiene un éxito del 90.8 % en textos formales y 45.8 % en tweets. En el contexto de los textos en tweets, existe una dificultad en el reconocimiento de entidades nombradas debido a la falta de información y datos de entrenamiento.

El trabajo principal de NER sobre tweets es Finin et al. [17], en donde se anotan los tweets y se entrena el modelo con CRF. En cuanto a los trabajos de NER sobre no tweets: Krupka y Hausman[35] utilizan reglas manuales para extraer entidades de tipos predefinidos, Zhou y Su [68] utilizan HMM (Hidden Markov Model) mientras que Finkel et al. [18] usa CRF. El trabajo más importante y actual de NER para tweets es Liu et al. [36] donde replantea el tema de reconocimiento de entidades nombradas en corpus de tweets y combina un clasificador KNN con CRF (Conditional Random Fields) para la detección.

La desambiguación léxica o etiquetado gramatical (POST) es una parte muy importante y útil en la tarea de normalización de textos ya que nos permite definir el

[4] http://aspell.net/

subconjunto de palabras debido a su categoría gramatical que con una probabilidad pueden ser la normalización de un OOV. Además un gran porcentaje de palabras en un texto son palabras que pueden ser asignadas a más de una clase morfológica, a más de un part-of-speech (PoS). Uno de los trabajos más importantes y probado para español es el de Sánchez-Villamil et al. [62], este trabajo presenta un método de POST de ventana deslizante (SWPoST), asigna el part-of-speech de una palabra basado en la información que dan las palabras en una ventana fija de alrededor. Este algoritmo puede ser implementado como una máquina de estados finitos (Máquina de Mealy).

2.4. Normalización en español

Una introducción a la normalización de tweets en español se presentó en la tarea Tweet Norm 2013 [3][2]. Este trabajo propuso en 2013 una tarea o competición en la que los participantes proponían soluciones de normalización de tweets. Los organizadores de la competición ofrecían dos datasets de tweets ya notados uno de desarrollo y otro para test, junto con un tercero que no era público y que era usado para la última evaluación.

Las soluciones ofrecidas por los participantes se pueden dividir en dos categorías, los que utilizan generación de candidatos junto un modelo del lenguaje, y los que utilizan transductores o FSTs (Finite State Transducers). El participante que mejor accurancy consiguió, Sistema RAE [22] con un 0.781, optó por la segunda categoría e implementó un siststem basado en FSTs para la tarea de normalización léxica de mensajes de Twitter en Español. El sistema desarrollado consiste en transductores que se aplican a tokens OOV. Los transductores implementan modelos de variación lingüística que generan conjuntos de candidatos acordes a un léxico. Un modelo estadístico del lenguaje se usa para obtener la secuencia de palabras más probable. El sistema tiene tres componentes principales que se aplican secuencialmente. Un analizador que ejecuta tokenización y análisis léxico sobre palabras en forma estándar y otras expresiones (números, fechas, ...). Un componente que genera palabras candidatas para los tokens OOV. Un modelo estadístico del lenguaje para obtener la mejor secuencia de palabras. Y finalmente un truecaser para capitalizar correctamente las palabras asignadas a los tokens OOV. El conjunto de confusión de un token OOV se genera aplicando el algoritmo de camino mínimo a la expresión: $W \circ E \circ L$. Donde W es el automata que representa el token OOV, E es un transductor de editado que genera todas las posibles variaciones de un token, y L es un conjunto de palabras objetivo. Dentro de esta categoría se encuentran los trabajos de la tarea: Ageno et al. [1] que usan una batería de módulos para generar diferentes propuestas de corrección para cada palabra desconocida. La corrección definitiva se elige por votación ponderada según la precisión de cada módulo, Alegria et al. [3] que además utilizan un modelo para el reconocimiento de voz para la generación de candidatos y Hulden y Francom [31] presentan dos estrategias basadas en FSTs una con reglas diseñadas manualmente y la otra automática.

Entre los participantes que optaron por la primera categoría destacan Ruiz et al. [54] con su sistema Vicomtech [5] que usan reglas de preproceso, un modelo de distancias de edición adecuado al dominio y modelos de lengua para seleccionar candidatos de corrección según el contexto. Su arquitectura está formada por: preproceso basado en expresiones regulares y listas customizadas, generación de candidatos mediante una técnica de mínima de distancia de editado, ranking de candidatos mediante una combinación con pesos de la puntuación del modelo del lenguaje y la distancia de editado y la puntuación del modelo de lenguaje es n-grama utilizando la distancia Levenshtein. El sistema obtuvo resultados superiores a la media en la tarea. En una mejora a este trabajo por los mismos autores utilizan un sistema basado en reglas para seleccionar los candidatos [55]. Otros trabajos en esta categoría son: Gamallo et al. [23] que propone un sistema basado en Han et al. [29], Saralegi y Sa Vicente Rocal [65] y Mosquera y Moreda[44] que emplean técnicas de RAH (reconocimiento del habla) mediante la herramienta TENOR presentada en Mosquera et al. [42] junto con un modelo del lenguaje. Otro trabajo basado en la tarea de Tweet-Norm pero que no participó en ella es el de Cerón-Guzmán y León-Guzmán[9], ellos optaron por normalizar los OOV basándose en similaridad entre grafemas y fonemas; generan el conjunto de confusión (de candidatos) usando grafemas y fonemas, seguido de transductores aplicados mediante reglas para las palabras extranjeras y acentos, la selección de candidatos mediante un modelo del lenguaje con la herramienta Kenlm [30].

Fuera de estas dos categorías nos encontramos con los trabajos: Montejo et al. [56] que utilizan conversiones basadas en reglas hasta una forma final normalizada. Después de recibir una lista con las posibles correcciones el sistema selecciona la más común acorde con una lista de palabras ordenada por frecuencia, Vilares et al. [66] utilizan una lista de prioridad para los candidatos obtenidos y una tabla de frencuencias de palabrar para puntuarlos, Han et al. [29] presentan una estrategia basada en búsquedas rápidas mediante una lista de frecuencias aprendida desde un corpus de tweets, Muñoz-García et al. [69] no generan candidatos simplemente seleccionan palabras OOV y las corrigen con un corrector externo y Cotelo et al. [45] generan candidatos y seleccionan el mejor mediante una función de distancia. En una mejora a este último trabajo por parte de los autores se añade un modulo de puntuación para la selección de candidatos [12].

Otros trabajos sobre normalización en español son Mosquera et al. [42] en donde se generan candidatos con indexación fonética y se seleccionan el candidato calculando la similaridad léxica junto con un modelo del lenguaje trigrama y Oliva et al. [48]. Estos trabajos son principalmente sobre mensajes SMS, y no abordan la normalización de tweets en su conjunto. Dentro de la normalización en español existen otras tareas relacionadas como es la tokenización y aquí destaca el trabajo Gomez-Hidalgo et al. [25] que estudia la tokenización de textos SMS.

[5] https://github.com/pruizf/tweet-norm-es

2.5. Word2Vec

Muchos sistemas y técnicas actuales de NLP tratan las palabras como unidades atómicas, no hay noción de similaridad entre palabras y son representadas como índices en un vocabulario, por ejemplo el modelo N-grama, para tratar de resolver este problema aparecen las representaciones continuas de palabras. Las representaciones continuas de palabras entrenadas sobre corpus sin etiquetas son útiles para muchos trabajos de NLP. Muchos tipos diferentes de modelos han sido propuestos para estimar representaciones continuas de palabras, incluyendo Latent Semantic Analysis (LSA) y Latent Dirichlet Allocation (LDA). En este trabajo se centran en las representaciones distribuidas de palabras aprendidas por redes neuronales, ya que se demostró que su eficacia era considerablemente mejor que LSA para preservar regularidades lineales entre palabras, LDA además es computacionalmente caro en datasets grandes. Además se ha demostrado las redes neuronales basadas en modelos del lenguaje mejoran significativamente los modelos N-grama [7] [37] [58].

Los dos modelos de redes neuronales que destacan basados en modelos del lenguaje son: Feedforward Neural Net Language Model (NNLM) [7] y Recurrent Neural Net Language Model (RNNLM) que mejora algunas limitaciones de NNLM. El problema de estos modelos es que con grandes cantidades de datos son muy costotos de obtener. Para resolver este problema en el trabajo de Mikolov et al. [38] desarrollado por Google se presentaron dos nuevos modelos de arquitecturas para calcular representaciones continuas de vectores de palabras a partir de grandes datasets, además crearon un framework de bibliotecas llamado Word2Vec [6]. El principal objetivo de este trabajo es introducir técnicas que puedan ser usadas para aprender vectores de palabras de gran calidad a partir de grandes datasets con millones de palabras y con millones de palabras en el vocabulario. Se trabajó con modelos más simples que aunque no puedan representar los datos de forma tan precisa como las redes neuronales pero pueden ser entrenados con muchos más datos de forma más eficiente. Estos modelos son: Continuous Baf-of-Words model (CBOW) similar a NNLM, la capa oculta no-lineal se elimina y la capa de proyección es compartida por todas las palabras; y Continuous Skip-gram model similar a CBOW pero en vez de predecir la palabra actual basándose en el contexto intenta maximizar la clasificación de la palabra basándose en otra palabra de la misma frase.

La mayoría de las técnicas de representación continua de vectores de palabras representan cada palabra del vocabulario como un vector distinto, sin parámetros compartidos. En particular se ignora la estructura interna de las palabras lo que es una importante limitación en lenguajes ricos morfológicamente. Para intentar resolver este problema en el trabajo Bojanowski et al. [51], desarrollado por Facebook y llamado fastText [7], se propone un nuevo enfoque basado en el modelo skipgram [38] donde cada palabra se representa como una bolsa de caracteres n-gramas. Una

[6] https://github.com/deeplearning4j/deeplearning4j
[7] https://fasttext.cc/

representación de vector está asociada con cada caracter n-grama, las palabras se representan como la suma de estas representaciones. Al usar una representación de vector distinta para cada palabra, el modelo skipgram ignora la estructura interna de las palabras y en este nuevo trabajo se implementa una función de puntuación diferente para tener en cuenta esta información.

3. Solución propuesta

La solución propuesta y a la que he llamado TweetSC (Tweet Spell Checker) [41] se llegó a ella a partir de varios análisis y evaluaciones que se hicieron con diversas bibliotecas y algoritmos, y todos ellos se pueden encontrar en la solución final para su uso.

En la primera versión de mi solución se construyó un corrector de texto sencillo basándonos en el creado por Peter Norvig [47], el cuál utiliza un diccionario para seleccionar las palabras incorrectas y las corrige mediante el teorema de Bayes usando probabilidades. Se usa la fórmula: $argmax_{c \in candidates} P(c|w)$, que mediante el teorema de Bayes es equivalente a: $argmax_{c \in candidates} P(c)P(w|c)/P(w)$, y como P(w) es igual para cada candidato c: $argmax_{c \in candidates} P(c)P(w|c)$. Esta fórmula trata de seleccionar el candidato de probabilidad máxima para cada palabra. Para calcular la probabilidad se usan dos diccionarios, uno de palabras en Español [8] y otro de nombres propios [9]. Se utilizó esta primera versión como punto de partida para ir creando versiones más avanzadas.

El resultado final y por tanto nuestra versión definitiva consiste en un proceso iterativo sobre el tweet que se puede dividir en 6 fases: Tokenización, reglas de preproceso, detección de OOVs, generación de candidatos para cada OOV, ranking de candidatos y postproceso.

Además para convertir el sistema en uno más dinámico se ha desarrollado una aplicación web con acceso a la API de Twitter para obtener los tweets mediante consultas introducidas en un formulario de nuestra aplicación web.

3.1. Tokenización

Cómo realizan los analizadores léxicos en los compiladores, en la primera fase de nuestro proceso se realiza una tokenización del texto o tweet.

Para mejorar la versión inicial se hizo uso de la biblioteca Stanford NLP [27]. Esta biblioteca creada por Stanford NLP Group ofrece tanto etiquetado gramatical (POS Tagging o POST) cómo deteccion de etiquetas (Named Entity Recognition o NER), yo la he utilizado para esta fase de tokenización. Además de StanfordNLP para la tokenización se ha utilizado Freeling [21], también se ha añadido al sistema el análisis que ofrece freeling.

En esta primera fase se recibe como entrada el texto del tweet y genera una lista de tokens que pasaran a la siguiente fase.

[8] https://github.com/jmorenov/TweetSC/blob/master/code/tweetsccore/src/main/resources/dic.txt

[9] https://github.com/jmorenov/TweetSC/blob/master/code/tweetsccore/src/main/resources/nombres_propios.txt

3.2. Reglas de preprocesado

Una vez que he obtenido todos los tokens de un tweet se aplican unas reglas de preproceso para normalizar palabras típicas de la red social, pictogramas, fonogramas, onomatopeyas, números, acrónimos, etc. Tras aplicar estas reglas a los tokens que las acepten, se crean OOVs con estos token, se anotan como variaciones y se eliminan de la lista de tokens para las fases siguientes. Los OOV generados se añaden a la lista final de OOV.

3.3. Detección de OOV

Esta fase tiene como elementos de entrada los tokens restantes de la fase anterior, y se ejecuta token por token el detector de OOV. Para detectarlos se aplican reglas y se van descartando los tokens que son URLs, usuarios de Twitter, *hashtags* de Twitter y fechas; los elementos restantes se comparan con tres diccionarios utilizados como recursos: diccionario de español, diccionario de inglés y diccionario de entidades.

Los token que se detecten dentro del diccionario de español se descartan como OOV, los que se detecten en el diccionario de inglés se anotan como NoEs (No español o ininteligible) y los que se detecten en el diccionario de entidades se anotan como Correct (palabras correspondientes a una entidad o un nuevo préstamo). Para el resto de token que no han sido aceptados en ninguna regla se crea una lista de OOV y son los que pasaran a la siguiente fase pudiendo al final ser anotados como Variation o NoEs.

3.4. Generación de candidatos OOV

La generación de candidatos se puede considerar la primera fase de la corrección en sí, ya que sólo se trabaja con OOV a los que se va a buscar una corrección, en ese caso candidatos para ese OOV. Esta fase tiene como entrada la lista de OOV que no han sido etiquetados en la fase anterior, es decir, los que pueden ser Variation o NoEs. Para cada OOV se generarán una lista de candidatos con diferentes métodos. Los métodos que he utilizado con los nombres que he definido son: LevenshteinFST, Metaphone, L_L, FastTest.

- **LevenshteinFST:** Método que utiliza un FST (Finite State Transducers) para generar variaciones en el OOV con un máximo de distancia de editado según la distancia Levenshtein.

- **Metaphone:** Método que utiliza el algoritmo del metáfono en español [43]. Su funcionamiento consiste en generar los fonemas de todos los diccionarios que he utilizado para después comparar el fonema del OOV y seleccionar los de mayor similaridad con los fonemas de los diccionarios.

- **L_L:** Los candidatos generados con este método son las palabras aceptadas por el lenguaje L(_L)+.
- **FastText:** Este método hace uso de la biblioteca fastText [16], a partir de un modelo generado mediante redes neuronales y representando las palabras de forma continua. Los OOV se convierten a vectores de palabras y se comparan con los vectores del modelo generado para obtener los candidatos más parecidos a partir del la similaridad del coseno.

Estos métodos se ejecutan sobre todos los OOV y generan una lista de candidatos que pasarán a la siguiente fase.

3.5. Ranking de candidatos

El ranking de candidatos es la fase que define la corrección de un OOV, o si no tiene corrección (se anota como NoEs). Para generar este ranking he utilizado dos marcadores, uno un modelo del lenguaje N-Gram (Modelo del lenguaje 11.1) mediante la biblioteca OpenNLP [4] y el otro la distancia de editado Damerau-Levenshtein.

El marcador N-Gram se realiza mediante la comparación de los candidatos de su puntuación en el modelo del lenguaje, es decir, para cada candidato se calcula su puntuación si fuera el elegido. Para el marcador de la distancia Damerau-Levenshtein se calculan la distancia entre el OOV y cada candidato. Finalmente mediante estas dos puntuaciones se calcula se realiza el ranking de candidatos posicionando primero los candidatos con mejor puntuación en el modelo del lenguaje y menor distancia de editado al OOV.

Se ha definido además un umbral mínimo para realizar el ranking, los candidatos que no lo cumplan con los marcadores son eliminados. Al finalizar este proceso para cada OOV se selecciona el mejor candidato del ranking, y se anota como Variation, y si no tuviera candidatos, debido al umbral, se anotan como NoEs.

3.6. Postproceso

Esta última fase consiste en poner mayúsculas en las palabras que fueran necesarias, así como signos de exclamación, interrogación y puntuación.

4. Implementación

En esta sección se pretende explicar la implementación software que se ha realizado de la solución. Mencionando los lenguajes y recursos externos utilizados, y explicando el funcionamiento de la aplicación web desarrollada. También se puede encontrar toda la documentación generada del proyecto al final de este documento (sección 9, sección 10, sección 11).

4.1. Diseño de arquitectura

La implementación se ha realizado en tres módulos o componentes. Por una parte tenemos la biblioteca con la funcionalidad necesaria para corregir textos de Twitter, acceder a su API y evaluar los resultados sobre un corpus de tweets; después un modulo que implementa aplicación web y por último otro que ofrece funcionalidad para utilizar la biblioteca desde línea de comandos.

El lenguaje principal utilizado en todo el proyecto ha sido Java, con la excepción de Python para los script de evaluación y normalización de archivos de datos, y he hecho uso de Google Cloud Engine [26] para que la aplicación web esté disponible para cualquier usuario [40].

Nuestro sistema software se ha intentado desarrollar de forma que sea un sistema de procesamiento dinámico pudiendo añadir y quitar funcionalidad de manera sencilla, cualquier algoritmo o método implementado funciona a partir de una clase superior para que se puedan añadir nuevos métodos.

El módulo principal del sistema es TweetSCCore que implementa la biblioteca para corregir tweets, acceder a la API de Twitter y evaluar resultados mediante el script ofrecido por Tweet-Norm 2013 [3]. Los módulos que funcionan a partir de TweetSCCore son: TweetSCExecutable que implementa la funcionalidad necesaria para ejectar nuestro sistema a partir de línea de comandos, y TweetSCWeb donde se implementa la aplicacion web.

Se han implementado dos métodos de corrección o normalización de tweets a los que he llamado DictionaryMethod, método preliminar que hace uso de diccionarios y la regla de Bayes, y TweetSCMethod que es nuestro método final con las fases que han sido explicadas.

Los módulos que forman el sistema son los siguientes:

- Tokenizer: Recibe el texto sin normalizar y lo tokeniza devolviendo una lista de tokens (sección 3.1).
- ApplyRules: Aplica las reglas de preprocesado a partir del archivo de reglas, recibe una lista de tokens y devuelve dos listas (OOV detectados con las reglas y tokens que no son OOV) (sección 3.2)

- OOVDetector: Detecta, a partir de una lista de tokens de entrada, palabras OOV utilizando los diccionarios de palabras en español, palabras en inglés y entidades. Además de detectar palabras OOV, las que son detectadas se anotan como se especifica en la tarea Tweet Norm 2013 [3]. Las palabras OOV que son anotadas como correctas o que no son en español pasan a la lista definitiva de OOV, el resto pasan al siguiente módulo (sección 3.3).
- Annotator: Forma parte del módulo OOVDetector y se utiliza para anotar las palabras OOV según la tarea Tweet Norm 2013 [3].
- CandidatesGenerator: Genera los candidatos para cada palabra OOV según los métodos que se definan: LevenshteinFST, Metaphone, L(_L)+ o fastText (sección 3.4).
- CandidatesRanking: Aplica un ranking a los candidatos de cada OOV para seleccionar el de mayor puntuación (sección 3.5).
- PostProceso: Último módulo en aplicarse del sistema, al texto final normalizado se le aplican reglas de postproceso para añadir signos de puntuación y mayúsculas (sección 3.6).

Todos estos módulos pertenecen al paquete tweetsccore [10].

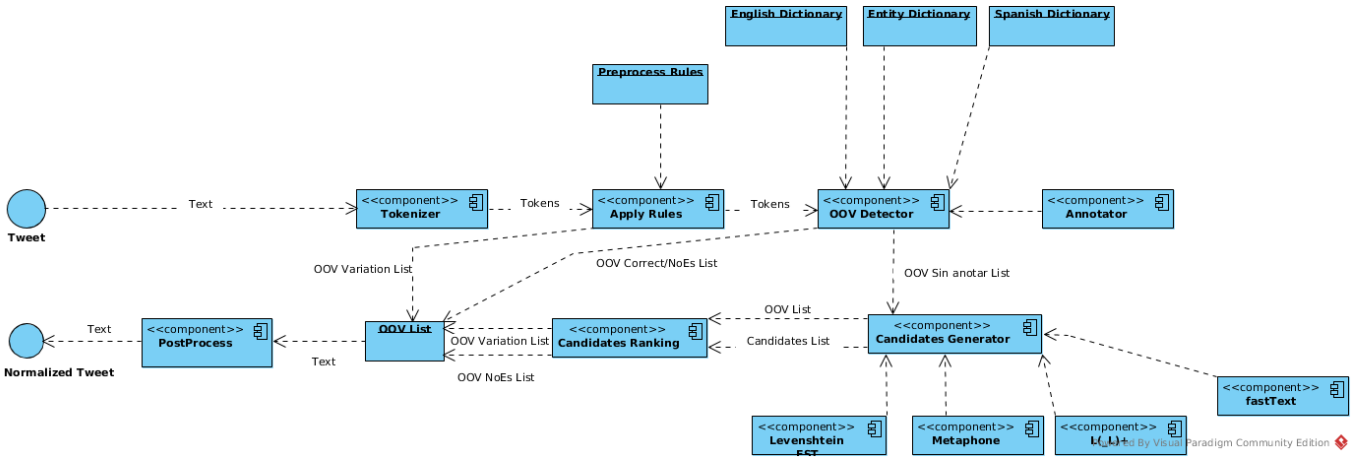

Fig. 2: Diagrama de módulos del sistema

[10] https://github.com/jmorenov/TweetSC/tree/master/code/tweetsccore

4.2. Instrucciones para usar el software

Para utilizar el software desarrollado primero es necesario tener instalado Git y Java 1.8. Después de bajar el código fuente:

```
git clone https://github.com/jmorenov/TweetSC
```

Posteriormente se compila el código:

```
cd TweetSC/code/
chmod +x build_all.sh
./build_all.sh
```

Para ejecutarlo desde línea de comandos:

```
java -jar tweetscexecutable-all-v0.5.0-alpha.jar -text Texto de prueba
```

La ejecución de la evaluación sobre el corpus de Tweet-Norm 2013 [3]

```
java -jar tweetscexecutable-all-v0.5.0-alpha.jar \
    -workingDirectory evaluation \
    -annotatedFile tweet-norm-dev500_annotated.txt \
    -tweetsFile tweet-norm-dev500.txt \
    -resultFile results-test-dev500.txt \
    -method TweetSCMethod
```

La ejecución de la aplicación web:

```
cd tweetscweb
./gradlew run
```

4.3. Aplicación web

En esta sección se explicará el funcionamiento de la aplicación web desarrollada y se mostrarán capturas de pantalla de la misma en funcionamiento.

La aplicación web se encuentra en nuestro paquete TweetSCWeb y hace uso del framework Spring Boot. Para el backend se utiliza Java y para el frontend Javascript (Jquery). Se ha intentado realizar un diseño sencillo y fluido usando la biblioteca Bootstrap.

El funcionamiento de la aplicación web es muy sencillo, tiene una sección principal desde la que se accede a las demás secciones mediante una barra de navegación.

Fig. 3: Inicio de la aplicación web

En la sección siguiente se puede ver un corrector de texto simple.

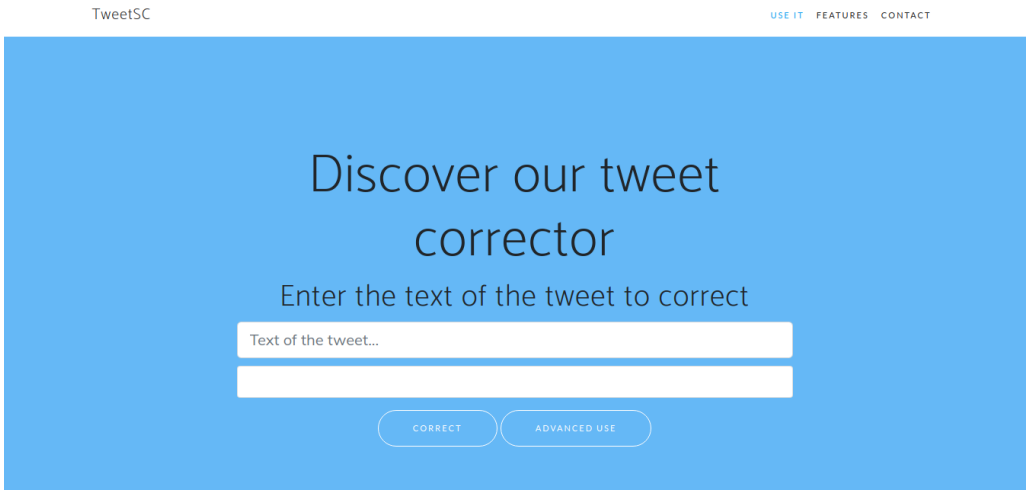

Fig. 4: Sección para utilizar el corrector

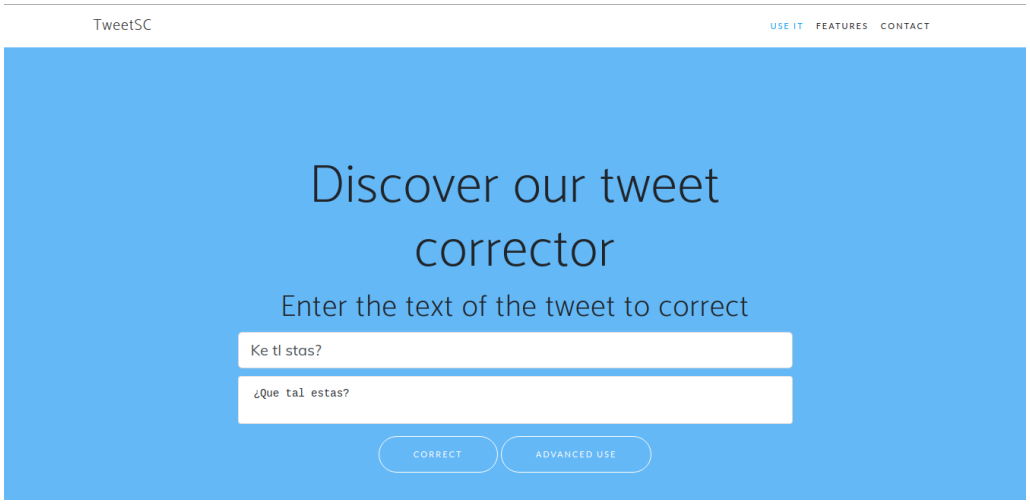

Fig. 5: Ejemplo de texto corregido

Si accedemos al corrector de uso avanzado podemos buscar tweets mediante texto, usuarios o id del tweet.

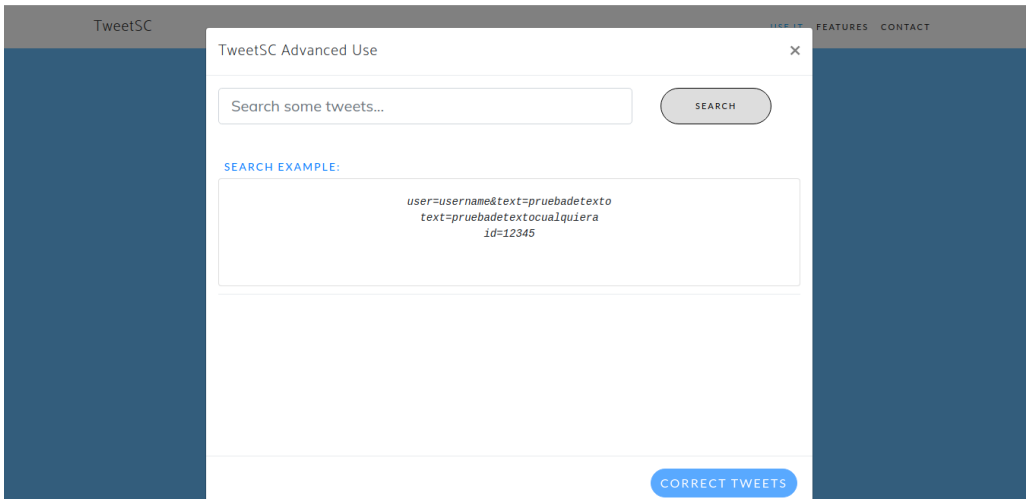

Fig. 6: Corrector de tweets uso avanzado

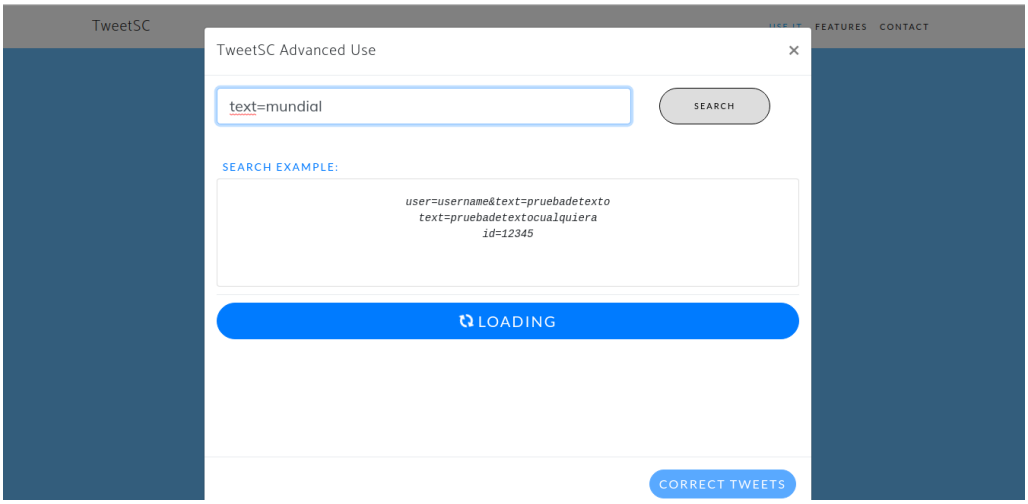

Fig. 7: Ejemplo de búsqueda de tweets

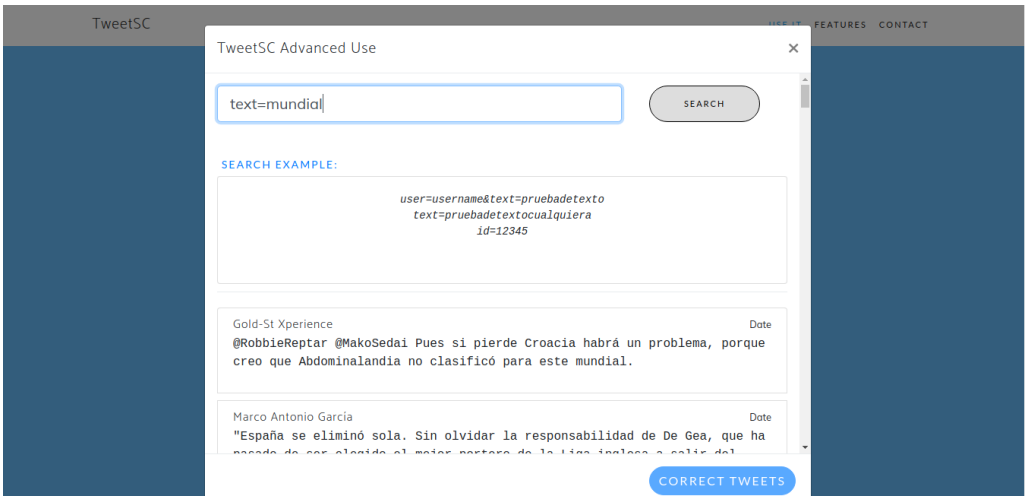

Fig. 8: Ejemplo de tweets encontrados

Los tweets encontrados se pueden seleccionar para corregirlos.

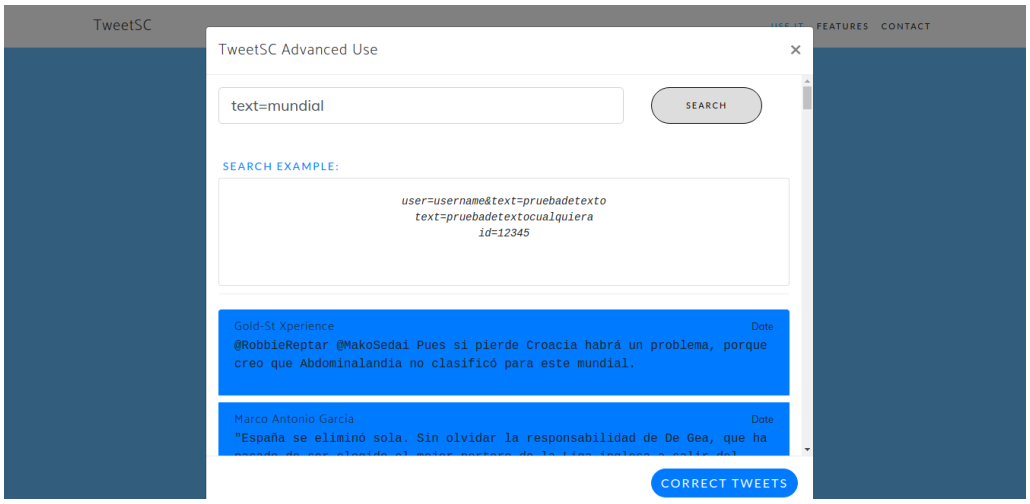

Fig. 9: Ejemplo de selección de tweets para corregir

Cuando los tweets son corregidos se marcan y se muestran ambas versiones, normalizada y la inicial.

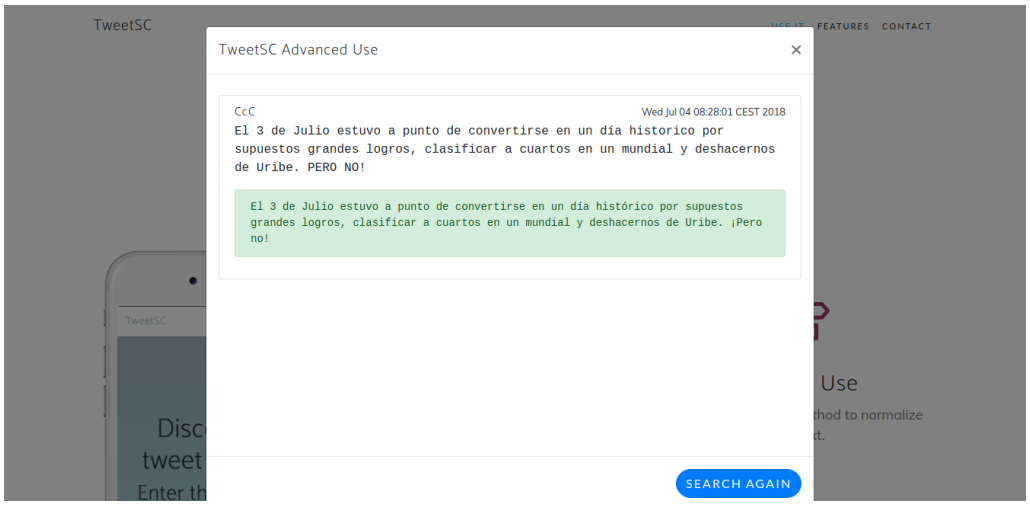

Fig. 10: Ejemplo de tweets corregidos

La siguiente sección es la de características del sistema.

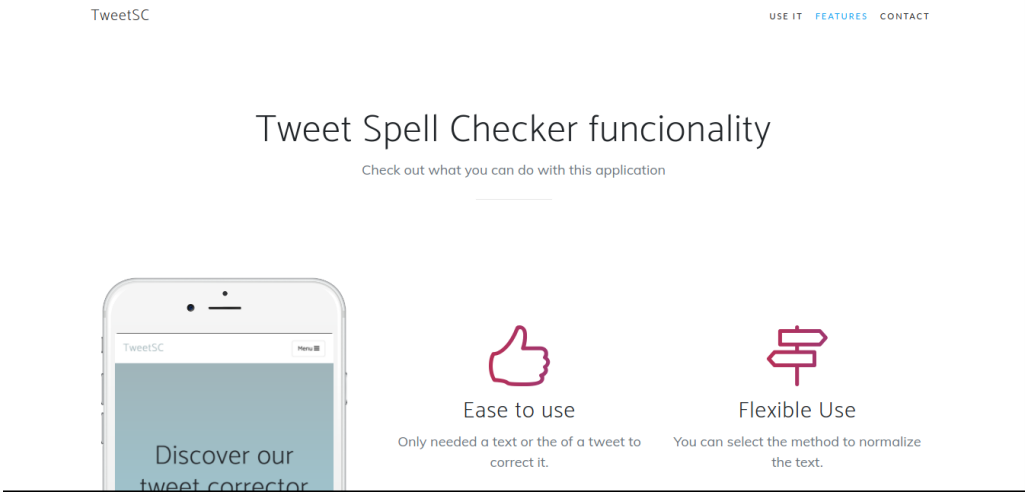

Fig. 11: Sección de características de la aplicación web

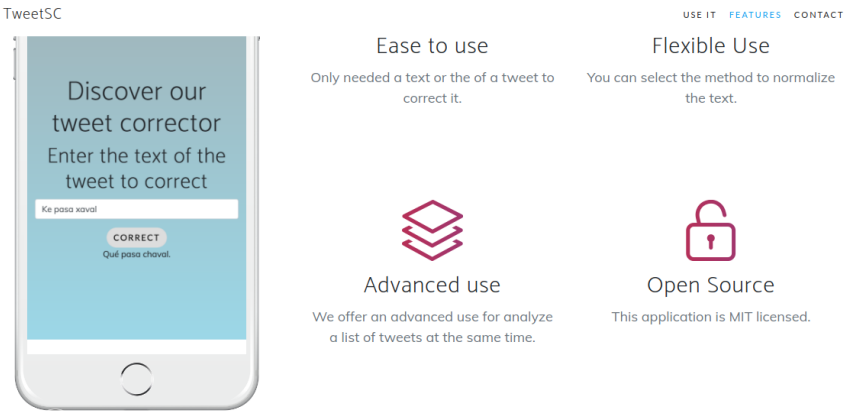

Fig. 12: Sección de características de la aplicación web

Además se ha añadido una sección para colaborar mediante GitHub en el desarrollo.

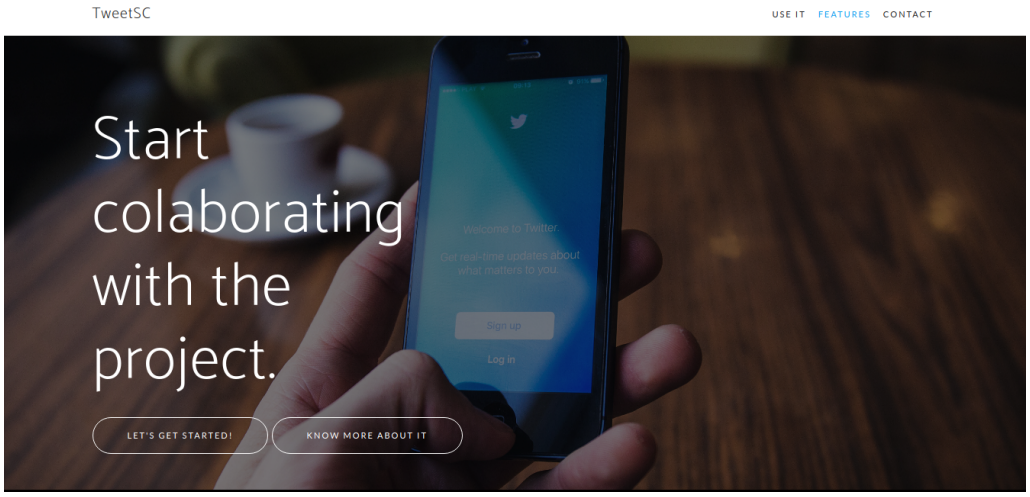

Fig. 13: Sección de colaboración

Por último la sección de contacto.

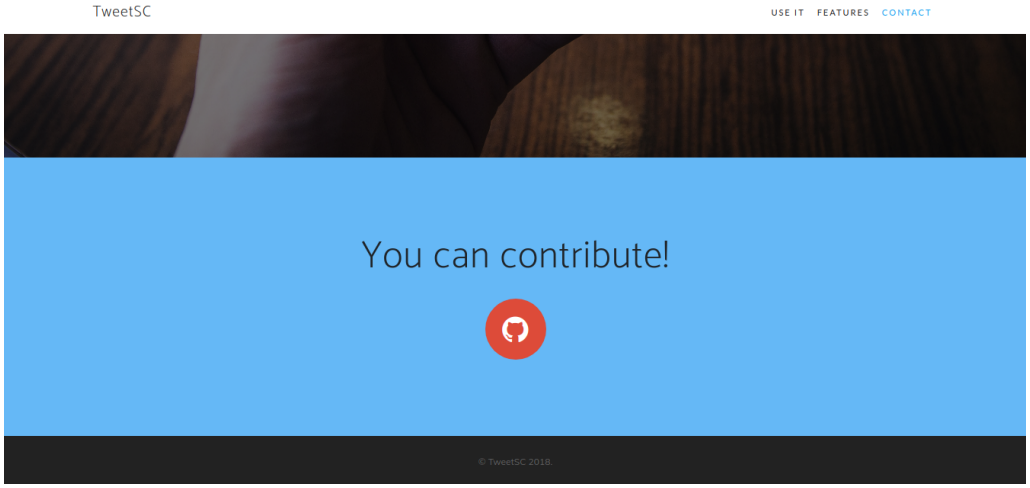

Fig. 14: Sección de contacto

5. Recursos utilizados

Los recursos utilizados por nuestro sistema son variados, desde diccionarios hasta bibliotecas para el desarrollo. Empezando desde el paquete TweetSCCore, se ha utilizado la biblioteca StanfordNLP [27] para la tokenización, además está disponible en el paquete la biblioteca Freeling [21]. El acceso a la API de Twitter se realiza mediante la bibliteca para Java Twitter4j. La detección de OOV utiliza tres diccionarios, el diccionario de español proporcionado por la herramienta Aspell, el diccionario de entidades JRC y un diccionario de inglés [13]. Los métodos de la generación de candidatos utilizan los recursos: Algoritmo del metáfono [43], fastText [16] y la biblioteca liblevenshtein [14] para el FST. El ranking de candidatos utiliza la biblioteca [4] para el modelo del lenguaje N-Grama y la distancia Damerau-Levenshtein ofrecida por String.Util de java.

Además el corpus de tweets para la evaluación es el que recopiló Tweet-Norm 2013 [3] y su script en Python para los resultados.

En la versión final se han utilizado cuatro diccionarios:

- Diccionario modificado y basado en GNU Aspell [11].
- Diccionario fonético de las palabras del diccionario anterior [12].
- Diccionario de palabras en inglés [13].
- Diccionario de entidades modificado y basado en JRC Entities [14].

También se ha utilizado un archivo de reglas de preproceso creado especificamente para este sistema [15].

El paquete TweetSCWeb utiliza el framework Spring Boot para la aplicación web junto con el sdk de Google Cloud para ofrecer la aplicación en la nube.

[11] https://github.com/jmorenov/TweetSC/blob/master/code/tweetsccore/src/main/resources/aspellNormalized.dict
[12] https://github.com/jmorenov/TweetSC/blob/master/code/tweetsccore/src/main/resources/aspellNormalizedPhonetic.dict
[13] https://github.com/jmorenov/TweetSC/blob/master/code/tweetsccore/src/main/resources/english.txt
[14] https://github.com/jmorenov/TweetSC/blob/master/code/tweetsccore/src/main/resources/jrc_entities.txt
[15] https://github.com/jmorenov/TweetSC/blob/master/code/tweetsccore/src/main/resources/preprocess/rules.txt

6. Evaluación

Esta sección describe la evaluación que se ha hecho de la solución propuesta. Primero se define la metodología utilizada para evaluar la solución, en segundo lugar explicamos el corpus utilizado como datos de entrada, posteriormente el gold standard actual y por último los experimentos que he realizada con sus resultados.

6.1. Metodología

La metodología que he seguido ha sido la misma que en la tarea compartida Tweet-Norm 2013 [64]. Ellos utilizan como medida de evaluación la corrección de errores, sólo tiene en cuenta si la forma propuesta es correcta en base a los criterios: **correcta** si la forma original era correcta y no se ha realizado ninguna normalización o si la forma original era incorrecta y el candidato seleccionado es el correcto; **errónea** en cualquier otro caso. La evaluación final es el número de decisiones realizadas correctamente sobre el total de palabras OOV.

6.2. Corpus

El corpus utilizado es el mismo que en la tarea compartida Tweet-Norm 2013 [64], en donde utilizan dos subconjuntos uno de desarrollo con 500 tweets y otro de evaluación con 600 tweets.

6.2.1. Gold Standard

Nuestro gold standard ha sido el sistema propuesto RAE [22] en Tweet-Norm 2013 [64] donde consiguieron un resultado de 0.781 de precisión. Su sistema se basa en trasductores de estados finitos con pesos.

6.3. Experimentos

Tab. 1: Resultados de ejecución de mi sistema (DictionaryMethod y TweetSCMethod) y comparación con resultados del sistema RAE [22] que obtuvo los mejores resultado en la tarea Tweet Norm 2013[3].

Método	**N**	**Positivos**	**Negativos**	***Accurancy***	**Tiempo de ej.**
DictionaryMethod	10	•	•	•	•
DictionaryMethod	100	•	•	•	•
DictionaryMethod	500	•	•	•	•
TweetSCMethod	10	•	•	•	•
TweetSCMethod	100	•	•	•	•
TweetSCMethod	500	•	•	•	•
Sistema RAE	?	•	•	•	•

Cómo se puede observar en la tabla 1 se muestran los resultados de mis dos métodos desarrollados, siendo DictionaryMethod la primera versión de mi sistema y consiste en un método básico de corrección con diccionarios, explicado en la sección 3. TweetSCMethod es la versión final de mi solución.

La ejecución se realiza sobre tres conjuntos de tweets: 10, 100 y 500. Los resultados que se ven en la tabla son: valores positivos, negativos, errores, *accurancy* y tiempo de ejecución.

7. Conclusiones y líneas futuras

Este Trabajo de Fin de Máster ha diseñado e implementado un corrector de textos en español en Twitter. El trabajo ha tenido una componente de investigación, y en algunos aspectos se ha ido más allá del estado arte, implementando las técnicas: Word2Vec (específicamente fastText), una FST con la distancia Levenshtein, el algoritmo del metáfono para realizar similaridad fonética entre palabras y un modelo del lenguaje N-grama. El sistema se ha implementado, es plenamente funcional y su código está accesible en https://github.com/jmorenov/TweetSC. Además, es posible verlo en acción en la web http://jmorenov.github.io/TweetSC/.

Se puede concluir que nuestro objetivo era construir un corrector de texto para Twitter (sección 1.2), juntos con los subobjetivos, y se ha conseguido cómo se ha demostrado en las secciones sección 3 y sección 4.

Los resultados en los experimentos han sido de menor calidad que el sistema RAE con el que se han comparado [22], un x % inferiores en *accurancy* con el conjunto de 500 tweets. Tras observar los errores detalladamente en los resultados he detectado que la mayoría de ellos proviene en la detección de OOV (sobretodo en palabras de habla inglesa o anglicismos) y en el ranking (la función de distancia y el modelo del lenguaje no tiene un funcionamiento óptimo).

Este sistema tiene proyección para mejorar estos resultados debido a que se ha diseñado de forma que sea posible añadir nuevos métodos tanto para generar candidatos cómo para realizar el ranking de los mismos o detectar palabras OOV. La primera mejora a realizar sería sobre el modelo del lenguaje N-grama para conseguir un buen funcionamiento del ranking.

Las líneas futuras son muy amplias ya que estamos en un tema bastante reciente, sobre todo en español cómo se puede ver en el sección 2, y los resultados se pueden mejorar de bastantes formas. Centrándonos en nuestro sistema una mejora futura sería el añadir contexto a los tweets a partir de sus *hashtags*, usuarios, imágenes o emoticonos; de forma que se pudiera reducir el conjunto de candidatos o añadir nuevos a partir de estos datos. También se podría realizar un análisis de sentimientos sobre el tweet después de normalizar por si se pudiera mejorar la corrección de algún OOV.

8. TweetSCCore Documentación del código (Java Documentation)

Class Hierarchy

Classes

- java.lang.Object
    - com.jmorenov.tweetsccore.analyzer.AnalysisElement (in 8.5.1, page 51)
    - com.jmorenov.tweetsccore.analyzer.Analyzer (in 8.5.7, page 53)
        - com.jmorenov.tweetsccore.analyzer.FreelingAnalyzer (in 8.5.14, page 54)
    - com.jmorenov.tweetsccore.candidates.Candidate (in 8.6.1, page 55)
    - com.jmorenov.tweetsccore.candidates.CandidatesGenerator (in 8.6.7, page 57)
    - com.jmorenov.tweetsccore.candidates.CandidatesMethod (in 8.6.13, page 58)
        - com.jmorenov.tweetsccore.candidates.FastTextCandidatesMethod (in 8.6.33, page 62)
        - com.jmorenov.tweetsccore.candidates.LevenshteinFSTCandidatesMethod (in 8.6.40, page 63)
        - com.jmorenov.tweetsccore.candidates.MetaphoneCandidatesMethod (in 8.6.47, page 64)
    - com.jmorenov.tweetsccore.candidates.CandidatesRanking (in 8.6.27, page 61)
    - com.jmorenov.tweetsccore.evaluation.TweetNormEvaluationResult (in 8.7.1, page 66)
    - com.jmorenov.tweetsccore.evaluation.TweetNormEvaluator (in 8.7.7, page 67)
    - com.jmorenov.tweetsccore.extra.File (in 8.8.8, page 72)
    - com.jmorenov.tweetsccore.extra.FreelingInitializator (in 8.8.14, page 74)
    - com.jmorenov.tweetsccore.extra.OOV (in 8.8.20, page 75)
    - com.jmorenov.tweetsccore.extra.OOVDetector (in 8.8.26, page 78)
    - com.jmorenov.tweetsccore.extra.Parser (in 8.8.32, page 79)
    - com.jmorenov.tweetsccore.extra.Token (in 8.8.38, page 82)
    - com.jmorenov.tweetsccore.method.Method (in 8.9.8, page 85)
        - com.jmorenov.tweetsccore.method.DictionaryMethod (in 8.9.1, page 84)
        - com.jmorenov.tweetsccore.method.TweetSCMethod (in 8.9.15, page 86)
    - com.jmorenov.tweetsccore.ner.NER (in 8.4.1, page 48)
        - com.jmorenov.tweetsccore.ner.StanfordNLPNER (in 8.4.14, page 50)
    - com.jmorenov.tweetsccore.ner.NERELement (in 8.4.8, page 48)
    - com.jmorenov.tweetsccore.post.FreeLingPOST (in 8.10.1, page 88)
    - com.jmorenov.tweetsccore.post.OpenNLPPOST (in 8.10.5, page 88)
    - com.jmorenov.tweetsccore.post.POST (in 8.10.9, page 89)
        - com.jmorenov.tweetsccore.post.StanfordNLPPOST (in 8.10.16, page 89)
    - com.jmorenov.tweetsccore.preprocess.ApplyRules (in 8.1.1, page 32)
    - com.jmorenov.tweetsccore.preprocess.ApplyRulesResult (in 8.1.7, page 33)
    - com.jmorenov.tweetsccore.preprocess.Rule (in 8.1.13, page 35)
    - com.jmorenov.tweetsccore.preprocess.Rules (in 8.1.19, page 36)

- com.jmorenov.tweetsccore.spellchecker.SpellChecker (in 8.11.1, page 90)
- com.jmorenov.tweetsccore.tokenizer.Tokenizer (in 8.12.22, page 95)
  - com.jmorenov.tweetsccore.tokenizer.FreelingTokenizer (in 8.12.1, page 92)
  - com.jmorenov.tweetsccore.tokenizer.OpenNLPTokenizer (in 8.12.8, page 93)
  - com.jmorenov.tweetsccore.tokenizer.StanfordNLPTokenizer (in 8.12.15, page 94)
- com.jmorenov.tweetsccore.twitter.Tweet (in 8.2.1, page 38)
  - com.jmorenov.tweetsccore.twitter.TweetCorrected (in 8.2.8, page 41)
- com.jmorenov.tweetsccore.twitter.TwitterConfiguration (in 8.2.15, page 45)
- com.jmorenov.tweetsccore.twitter.api.Search (in 8.3.1, page 45)
- java.lang.Enum
  - com.jmorenov.tweetsccore.candidates.CandidatesMethodType (in 8.6.20, page 59)
  - com.jmorenov.tweetsccore.extra.Annotation (in 8.8.1, page 71)

8.1. Package com.jmorenov.tweetsccore.preprocess

Package Contents *Page*

Classes

8.1.1. Class ApplyRules

ApplyRules class to apply preprocess rules.

8.1.2. Declaration

```
public class ApplyRules
 extends java.lang.Object
```

8.1.3. Constructor summary

ApplyRules() Constructor of the class.

8.1.4. Method summary

apply(List) Method to apply the rules to a List of Token.
apply(String) Method to apply the rules to a text.

8.1.5. Constructors

- **ApplyRules**

```
public ApplyRules()
```

  - **Description**
    Constructor of the class.

8.1.6. Methods

- **apply**

```
public ApplyRulesResult apply(java.util.List tokens)
```

  - **Description**
    Method to apply the rules to a List of Token.
  - **Parameters**
    - `tokens` – List of tokens
  - **Returns** – ApplyRulesResult

- **apply**

```
public java.util.List apply(java.lang.String text)
```

  - **Description**
    Method to apply the rules to a text.
  - **Parameters**
    - `text` – String with the text
  - **Returns** – List of OOVs

8.1.7. Class ApplyRulesResult

8.1.8. Declaration

```
public class ApplyRulesResult
 extends java.lang.Object
```

8.1.9. Constructor summary

ApplyRulesResult() Constructor of the class.

8.1.10. Method summary

addOOV(OOV) Method to add an OOV to the list.
addToken(Token) Method to add a token to the list.
getOOVList() Method to get the list of OOVs.
getTokens() Method to get the list of tokens.
setOOVList(List) Method to set the list of OOV.
setTokens(List) Method to set the list of tokens.

8.1.11. Constructors

- **ApplyRulesResult**

```
public ApplyRulesResult()
```

  - **Description**
    Constructor of the class.

8.1.12. Methods

- **addOOV**

```
public void addOOV(com.jmorenov.tweetsccore.extra.OOV
    oov)
```

  - **Description**
    Method to add an OOV to the list.
  - **Parameters**
    - `oov` – OOV

- **addToken**

```
public void addToken(com.jmorenov.tweetsccore.extra.
    Token token)
```

  - **Description**
    Method to add a token to the list.
  - **Parameters**
    - `token` – Token

- **getOOVList**

```
public java.util.List getOOVList()
```

- **Description**
  Method to get the list of OOVs.
- **Returns** – List OOV

- **getTokens**

```
public java.util.List getTokens()
```

- **Description**
  Method to get the list of tokens.
- **Returns** – List Token

- **setOOVList**

```
public void setOOVList(java.util.List oov)
```

- **Description**
  Method to set the list of OOV.
- **Parameters**
  - `oov` – List OOV

- **setTokens**

```
public void setTokens(java.util.List tokens)
```

- **Description**
  Method to set the list of tokens.
- **Parameters**
  - `tokens` – List Token

8.1.13. Class Rule

Rule class that define a rule element.

8.1.14. Declaration

```
public class Rule
 extends java.lang.Object
```

8.1.15. Constructor summary

Rule(String, String) Constructor of the class.

8.1.16. Method summary

getRegex() Method to get the regex of the rule.
getResult() Method to get the result of a rule.

8.1.17. Constructors

- **Rule**

```
public Rule(java.lang.String regex,java.lang.String
    result)
```

  - **Description**
    Constructor of the class.
  - **Parameters**
    - `regex` – String
    - `result` – String

8.1.18. Methods

- **getRegex**

```
public java.lang.String getRegex()
```

  - **Description**
    Method to get the regex of the rule.
  - **Returns** – String

- **getResult**

```
public java.lang.String getResult()
```

  - **Description**
    Method to get the result of a rule.
  - **Returns** – String

8.1.19. Class Rules

Rule class that define the Rules element.

8.1.20. Declaration

```
public class Rules
 extends java.lang.Object
```

8.1.21. Constructor summary

Rules(String) Constructor of the class.

8.1.22. Method summary

addRule(Rule) Method to add a new rule.
getRules() Method to get the rules.

8.1.23. Constructors

- **Rules**

```
public Rules(java.lang.String rulesFilename) throws java
    .io.IOException
```

  - **Description**
    Constructor of the class.
  - **Parameters**
    - `rulesFilename` – String with the file name of the rules
  - **Throws**
    - `java.io.IOException` – When the file is not found

8.1.24. Methods

- **addRule**

```
public void addRule(Rule rule)
```

  - **Description**
    Method to add a new rule.
  - **Parameters**
    - `rule` – Rule

- **getRules**

```
public java.util.List getRules()
```

- **Description**
  Method to get the rules.
- **Returns** – List of Rule

8.2. Package com.jmorenov.tweetsccore.twitter

Package Contents *Page*

Classes

8.2.1. Class Tweet

Tweet class with the structure of a tweet.

8.2.2. Declaration

```
public class Tweet
 extends java.lang.Object
```

8.2.3. All known subclasses

TweetCorrected (in 8.2.8, page 41)

8.2.4. Constructor summary

Tweet() Default constructor of the class.
Tweet(Status) Constructor of the class.
Tweet(String, String, String, String) Constructor of the class.
Tweet(String, String, String, String, String) Constructor of the class.
Tweet(Tweet) Copy constructor

8.2.5. Method summary

getDate() Method to get the date of the tweet.
getHash() Method to get the hash of the tweet.
getId() Method to get the id of the tweet.
getText() Method to get the text of the tweet.

getUsername() Method to get the username of the tweet.
toString() Method to get the string of the Tweet.

8.2.6. Constructors

- **Tweet**

```
public Tweet()
```

  - **Description**
    Default constructor of the class.

- **Tweet**

```
public Tweet(Status tweetStatus)
```

  - **Description**
    Constructor of the class.
  - **Parameters**
    - `tweetStatus` – Status from the object of Twitter4j.

- **Tweet**

```
public Tweet(java.lang.String id,java.lang.String
    username,java.lang.String hash,java.lang.String text)
```

  - **Description**
    Constructor of the class.
  - **Parameters**
    - `id` – String with the id of the tweet.
    - `username` – String with the username of the tweet.
    - `hash` – String with the hash of the tweet.
    - `text` – String with the text of the tweet.

- **Tweet**

```
public Tweet(java.lang.String id,java.lang.String
    username,java.lang.String hash,java.lang.String text,
    java.lang.String date)
```

- **Description**
  Constructor of the class.
- **Parameters**
  - `id` – String with the id of the tweet.
  - `username` – String with the username of the tweet.
  - `hash` – String with the hash of the tweet.
  - `text` – String with the text of the tweet.
  - `date` – String with the date of the tweet.

- **Tweet**

```
public Tweet(Tweet tweet)
```

  - **Description**
    Copy constructor
  - **Parameters**
    - `tweet` – Tweet to copy from.

8.2.7. Methods

- **getDate**

```
public java.lang.String getDate()
```

  - **Description**
    Method to get the date of the tweet.
  - **Returns** – String with the date of the tweet.

- **getHash**

```
public java.lang.String getHash()
```

  - **Description**
    Method to get the hash of the tweet.
  - **Returns** – String with the hash of the tweet.

- **getId**

```
public java.lang.String getId()
```

- **Description**
  Method to get the id of the tweet.
- **Returns** – String with the id of the tweet.

▪ **getText**

```
public java.lang.String getText()
```

- **Description**
  Method to get the text of the tweet.
- **Returns** – String with the text of the tweet.

▪ **getUsername**

```
public java.lang.String getUsername()
```

- **Description**
  Method to get the username of the tweet.
- **Returns** – String with the username of the tweet.

▪ **toString**

```
public java.lang.String toString()
```

- **Description**
  Method to get the string of the Tweet.
- **Returns** – String with the String of the Tweet.

8.2.8. Class TweetCorrected

Tweet corrected class with the structure of a corrected tweet.

8.2.9. Declaration

```
public class TweetCorrected
 extends com.jmorenov.tweetsccore.twitter.Tweet
```

8.2.10. Constructor summary

TweetCorrected() Default constuctor of the class.
TweetCorrected(String) Constructor of the class.
TweetCorrected(String, String, String, String, String) Constructor of the class.
TweetCorrected(Tweet) Constructor from tweet.

8.2.11. Method summary

computeCorrectedText() Method to set the corrected text from the OOV words.
getCorrectedText() Method to get the corrected tweet.
getOOVWords() Method to get the Out-Of-Vocabulary words of the tweet.
setCorrectedText(String) Method to set the corrected tweet.
setOOVWords(List) Method to set the Out-Of-Vocabulary words of the tweet.
toString() Method to get the string of the Tweet Corrected.
toTweetNormString() Method to get the corrected text for Tweet Norm 2013.

8.2.12. Constructors

- **TweetCorrected**

```
public TweetCorrected()
```

  - **Description**
    Default constuctor of the class.

- **TweetCorrected**

```
public TweetCorrected(java.lang.String text)
```

  - **Description**
    Constructor of the class.
  - **Parameters**
    - `text` – of the tweet

- **TweetCorrected**

```
public TweetCorrected(java.lang.String id,java.lang.
    String username,java.lang.String hash,java.lang.
    String text,java.lang.String date)
```

  - **Description**
    Constructor of the class.
  - **Parameters**
    - `id` – of the tweet

- `username` –
- `hash` – of the tweet
- `text` – of the tweet
- `date` – of the tweet

- **TweetCorrected**

```
public TweetCorrected(Tweet tweet)
```

  - **Description**
    Constructor from tweet.
  - **Parameters**
    - `tweet` – Tweet

8.2.13. Methods

- **computeCorrectedText**

```
public void computeCorrectedText()
```

  - **Description**
    Method to set the corrected text from the OOV words.

- **getCorrectedText**

```
public java.lang.String getCorrectedText()
```

  - **Description**
    Method to get the corrected tweet.
  - **Returns** – String the corrected text

- **getOOVWords**

```
public java.util.List getOOVWords()
```

  - **Description**
    Method to get the Out-Of-Vocabulary words of the tweet.
  - **Returns** – List of OOV

- **setCorrectedText**

```
public void setCorrectedText(java.lang.String
    correctedText)
```

- **Description**
  Method to set the corrected tweet.
- **Parameters**
  - `correctedText` – the corrected text

- **setOOVWords**

```
public void setOOVWords(java.util.List OOVWords)
```

- **Description**
  Method to set the Out-Of-Vocabulary words of the tweet.
- **Parameters**
  - `OOVWords` – the list of OOV

- **toString**

```
public java.lang.String toString()
```

- **Description**
  Method to get the string of the Tweet Corrected.
- **Returns** – String with the String of the class

- **toTweetNormString**

```
public java.lang.String toTweetNormString()
```

- **Description**
  Method to get the corrected text for Tweet Norm 2013.
- **Returns** – String with the correctec text.

8.2.14. Members inherited from class Tweet

`com.jmorenov.tweetsccore.twitter.Tweet` (in 8.2.1, page 38)

- `public String` **`getDate()`**
- `public String` **`getHash()`**
- `public String` **`getId()`**
- `public String` **`getText()`**
- `public String` **`getUsername()`**
- `public String` **`toString()`**

8.2.15. Class TwitterConfiguration

TwitterConfiguration class with the configuration of the connection with the API of twitter.

8.2.16. Declaration

```
public class TwitterConfiguration
 extends java.lang.Object
```

8.2.17. Method summary

getInstance() Method to get the instance of the class.
getTwitterAccess() Method to get the access to the API.

8.2.18. Methods

- **getInstance**

  ```
  public static TwitterConfiguration getInstance()
  ```

  - **Description**
    Method to get the instance of the class.
  - **Returns** – TwitterConfiguration the instance of the class

- **getTwitterAccess**

  ```
  public Twitter getTwitterAccess()
  ```

  - **Description**
    Method to get the access to the API.
  - **Returns** – Twitter

8.3. Package com.jmorenov.tweetsccore.twitter.api

Package Contents *Page*

Classes

8.3.1. Class Search

Search class to do call on the Twitter API about tweets.

8.3.2. Declaration

```
public class Search
 extends java.lang.Object
```

8.3.3. Constructor summary

Search() Default constructor of the class.

8.3.4. Method summary

getAllTweetsOfUser(String) Method to get all the tweets of an user.
getTweetById(String) Method to search tweets by it id.
getTweetsByText(String) Method to search tweets.
getTweetsByTextOfUser(String, String) Method to search tweets of an user.

8.3.5. Constructors

- **Search**

  ```
  public Search()
  ```

  - **Description**
    Default constructor of the class.

8.3.6. Methods

- **getAllTweetsOfUser**

  ```
  public java.util.List getAllTweetsOfUser(java.lang.
      String username)
  ```

  - **Description**
    Method to get all the tweets of an user.
  - **Parameters**
    - `username` – the user
  - **Returns** – List of Status with the tweets

- **getTweetById**

  ```
  public com.jmorenov.tweetsccore.twitter.Tweet
      getTweetById(java.lang.String id)
  ```

- **Description**
  Method to search tweets by it id.
- **Parameters**
  - `id` – the id of the tweet
- **Returns** – Status The tweet

▪ **getTweetsByText**

```
public java.util.List getTweetsByText(java.lang.String
    text)
```

- **Description**
  Method to search tweets.
- **Parameters**
  - `text` – the text to search
- **Returns** – List of Status with the tweets

▪ **getTweetsByTextOfUser**

```
public java.util.List getTweetsByTextOfUser(java.lang.
    String username,java.lang.String text)
```

- **Description**
  Method to search tweets of an user.
- **Parameters**
  - `username` – the user
  - `text` – the text to search
- **Returns** – List of Status with the tweets

8.4. Package com.jmorenov.tweetsccore.ner

Package Contents *Page*

Classes

8.4.1. Class NER

POSTagging abstract class.

8.4.2. Declaration

```
public abstract class NER
 extends java.lang.Object
```

8.4.3. All known subclasses

StanfordNLPNER (in 8.4.14, page 50)

8.4.4. Constructor summary

NER()

8.4.5. Method summary

getNERElements(String) Method to get a list with the NER Elements detected.

8.4.6. Constructors

- **NER**

```
public NER()
```

8.4.7. Methods

- **getNERElements**

```
public abstract java.util.List getNERElements(java.lang.
    String text)
```

  - **Description**
    Method to get a list with the NER Elements detected.
  - **Returns** – List with the NER Elements.

8.4.8. Class NERELement

NERELement class.

8.4.9. Declaration

```
public class NERELement
 extends java.lang.Object
```

8.4.10. Constructor summary

NERELement(String, String) Constructor of the class

8.4.11. Method summary

getNerDetected() Method to get the ner detected.
getOriginalElement() Method to get the original element.

8.4.12. Constructors

- **NERELement**

  ```
  public NERELement(java.lang.String originalElement,java.
      lang.String nerDetected)
  ```

  - **Description**
    Constructor of the class
  - **Parameters**
    - `originalElement` –
    - `nerDetected` –

8.4.13. Methods

- **getNerDetected**

  ```
  public java.lang.String getNerDetected()
  ```

  - **Description**
    Method to get the ner detected.
  - **Returns** – String with the ner detected

- **getOriginalElement**

  ```
  public java.lang.String getOriginalElement()
  ```

  - **Description**
    Method to get the original element.
  - **Returns** – String with the original element

8.4.14. Class StanfordNLPNER

StanfordNLP class.

8.4.15. Declaration

```
public class StanfordNLPNER
 extends com.jmorenov.tweetsccore.ner.NER
```

8.4.16. Constructor summary

StanfordNLPNER(String)

8.4.17. Method summary

getNERElements(String) Method to get a list with the NER Elements detected.

8.4.18. Constructors

- **StanfordNLPNER**

```
public StanfordNLPNER(java.lang.String text)
```

8.4.19. Methods

- **getNERElements**

```
public java.util.List getNERElements(java.lang.String
    text)
```

  - **Description**
    Method to get a list with the NER Elements detected.
  - **Returns** – List with the NER Elements.

8.4.20. Members inherited from class NER

`com.jmorenov.tweetsccore.ner.NER` (in 8.4.1, page 48)

- `public abstract List` **`getNERElements`**`(java.lang.String` **`text`**`)`

8.5. Package com.jmorenov.tweetsccore.analyzer

Package Contents *Page*

Classes

8.5.1. Class AnalysisElement

AnalysisElement class.

8.5.2. Declaration

```
public class AnalysisElement
 extends java.lang.Object
```

8.5.3. Constructor summary

AnalysisElement(String, String, String, String, boolean) Constructor of the class.

8.5.4. Method summary

getForm() Method to get the form.
getLemma() Method to get the lemma.
getSenses() Method to get the senses.
getTag() Method to get the tag.
isMultiWord() Method to get if the element is multi word.
toString() Method to get the class as String.

8.5.5. Constructors

- **AnalysisElement**

```
public AnalysisElement(java.lang.String form,java.lang.
    String lemma,java.lang.String tag,java.lang.String
    senses,boolean isMultiWord)
```

  - **Description**
    Constructor of the class.

- **Parameters**
  - `form` –
  - `lemma` –
  - `tag` –
  - `isMultiWord` –

8.5.6. Methods

- **getForm**

  ```
  public java.lang.String getForm()
  ```

  - **Description**
    Method to get the form.
  - **Returns** – String with the form

- **getLemma**

  ```
  public java.lang.String getLemma()
  ```

  - **Description**
    Method to get the lemma.
  - **Returns** – String with the lemma

- **getSenses**

  ```
  public java.lang.String getSenses()
  ```

  - **Description**
    Method to get the senses.
  - **Returns** – String with the senses

- **getTag**

  ```
  public java.lang.String getTag()
  ```

  - **Description**
    Method to get the tag.
  - **Returns** – String with the tag

- **isMultiWord**

```
public boolean isMultiWord()
```

  - **Description**
    Method to get if the element is multi word.
  - **Returns** – Boolean with the test

- **toString**

```
public java.lang.String toString()
```

  - **Description**
    Method to get the class as String.
  - **Returns** – String

8.5.7. Class Analyzer

Analyzer abstract class.

8.5.8. Declaration

```
public abstract class Analyzer
 extends java.lang.Object
```

8.5.9. All known subclasses

FreelingAnalyzer (in 8.5.14, page 54)

8.5.10. Constructor summary

Analyzer()

8.5.11. Method summary

analyzeText(String) Method to get a list with the Analysis Elements.

8.5.12. Constructors

- **Analyzer**

```
public Analyzer()
```

8.5.13. Methods

- **analyzeText**

```
public abstract java.util.List analyzeText(java.lang.
    String text)
```

  - **Description**
    Method to get a list with the Analysis Elements.
  - **Returns** – List with the Analysis Elements.

8.5.14. Class FreelingAnalyzer

FreelingTokenizer class to tokenize a text.

8.5.15. Declaration

```
public class FreelingAnalyzer
 extends com.jmorenov.tweetsccore.analyzer.Analyzer
```

8.5.16. Constructor summary

FreelingAnalyzer() Constructor of the class.

8.5.17. Method summary

analyzeText(String) Method to get a list with the NER Elements detected.

8.5.18. Constructors

- **FreelingAnalyzer**

```
public FreelingAnalyzer()
```

  - **Description**
    Constructor of the class.

8.5.19. Methods

- **analyzeText**

```
public java.util.List analyzeText(java.lang.String text)
```

- **Description**
  Method to get a list with the NER Elements detected.
- **Returns** – List with the NER Elements.

8.5.20. Members inherited from class Analyzer

`com.jmorenov.tweetsccore.analyzer.Analyzer` (in 8.5.7, page 53)

- `public abstract List` **`analyzeText`**`(java.lang.String text)`

8.6. Package com.jmorenov.tweetsccore.candidates

Package Contents *Page*

Classes

8.6.1. Class Candidate

Candidate class that define the candidate element.

8.6.2. Declaration

```
public class Candidate
 extends java.lang.Object implements java.lang.Comparable
```

8.6.3. Constructor summary

Candidate(String, CandidatesMethodType) Constructor of the class.

8.6.4. Method summary

compareTo(Candidate) Method to compare two candidates.
getCandidate() Method to get the candidate.
getGeneratedBy() Method to get the method that generated the candidate.
getScore() Method to get the score of the candidate.
setScore(double) Method to set the score of the candidate.

8.6.5. Constructors

- **Candidate**

```
public Candidate(java.lang.String candidate,
    CandidatesMethodType generatedBy)
```

  - **Description**
    Constructor of the class.
  - **Parameters**
    - `candidate` – String with the candidate
    - `generatedBy` – String with the method that generated the candidate

8.6.6. Methods

- **compareTo**

```
public int compareTo(Candidate candidate)
```

  - **Description**
    Method to compare two candidates.
  - **Parameters**
    - `candidate` – Candidate
  - **Returns** – int

- **getCandidate**

```
public java.lang.String getCandidate()
```

- **Description**
  Method to get the candidate.
- **Returns** – String with the candidate

▪ **getGeneratedBy**

```
public CandidatesMethodType getGeneratedBy()
```

- **Description**
  Method to get the method that generated the candidate.
- **Returns** – CandidatesMethodType with the generated by

▪ **getScore**

```
public double getScore()
```

- **Description**
  Method to get the score of the candidate.
- **Returns** – double

▪ **setScore**

```
public void setScore(double score)
```

- **Description**
  Method to set the score of the candidate.
- **Parameters**
  - `score` – double

8.6.7. Class CandidatesGenerator

CandidatesGenerator class that define the generator of candidates.

8.6.8. Declaration

```
public class CandidatesGenerator
 extends java.lang.Object
```

8.6.9. Constructor summary

CandidatesGenerator(List) Constructor of the class.

8.6.10. Method summary

generateCandidates(List) Method to generate the candidates of an OOV.

8.6.11. Constructors

- **CandidatesGenerator**

```
public CandidatesGenerator(java.util.List methods)
```

  - **Description**
    Constructor of the class.
  - **Parameters**
    - `methods` – List of CandidatesMethod

8.6.12. Methods

- **generateCandidates**

```
public java.util.List generateCandidates(java.util.List
    oovs)
```

  - **Description**
    Method to generate the candidates of an OOV.
  - **Parameters**
    - `oovs` – List of OOV
  - **Returns** – List of OOV

8.6.13. Class CandidatesMethod

CandidatesMethod abstract class that define a method to generate candidates.

8.6.14. Declaration

```
public abstract class CandidatesMethod
 extends java.lang.Object
```

8.6.15. All known subclasses

MetaphoneCandidatesMethod (in 8.6.47, page 64), FastTextCandidatesMethod (in 8.6.33, page 62), LevenshteinFSTCandidatesMethod (in 8.6.40, page 63)

8.6.16. Constructor summary

CandidatesMethod()

8.6.17. Method summary

generateCandidates(OOV) Abstract method to generate condidates from an OOV.
getMethod() Abstract method to obtain the method description.

8.6.18. Constructors

- **CandidatesMethod**

```
public CandidatesMethod()
```

8.6.19. Methods

- **generateCandidates**

```
public abstract java.util.List generateCandidates(com.
    jmorenov.tweetsccore.extra.OOV oov)
```

  - **Description**
    Abstract method to generate condidates from an OOV.
  - **Parameters**
    - `oov` – OOV
  - **Returns** – List of Candidates

- **getMethod**

```
public abstract CandidatesMethodType getMethod()
```

  - **Description**
    Abstract method to obtain the method description.
  - **Returns** – CandidatesMethodType

8.6.20. Class CandidatesMethodType

CandidatesMethodType enum with the different methods to generate candidates.

8.6.21. Declaration

```
public final class CandidatesMethodType
 extends java.lang.Enum
```

8.6.22. Field summary

FastText
L_L
LevenshteinFST
Metaphone

8.6.23. Method summary

valueOf(String)
values()

8.6.24. Fields

- `public static final CandidatesMethodType` **LevenshteinFST**
- `public static final CandidatesMethodType` **Metaphone**
- `public static final CandidatesMethodType` **L_L**
- `public static final CandidatesMethodType` **FastText**

8.6.25. Methods

- **valueOf**

  ```
  public static CandidatesMethodType valueOf(java.lang.
     String name)
  ```

- **values**

  ```
  public static CandidatesMethodType[] values()
  ```

8.6.26. Members inherited from class Enum

`java.lang.Enum`

- `protected final Object` **clone**`() throws CloneNotSupportedException`
- `public final int` **compareTo**`(Enum` **arg0**`)`
- `public final boolean` **equals**`(Object` **arg0**`)`
- `protected final void` **finalize**`()`
- `public final Class` **getDeclaringClass**`()`

- `public final int` **hashCode**`()`
- `public final String` **name**`()`
- `public final int` **ordinal**`()`
- `public String` **toString**`()`
- `public static Enum` **valueOf**`(Class` **arg0**`, String` **arg1**`)`

8.6.27. Class CandidatesRanking

CandidatesRanking class to ranking the candidates.

8.6.28. Declaration

```
public class CandidatesRanking
 extends java.lang.Object
```

8.6.29. Constructor summary

CandidatesRanking() Constructor of the class.

8.6.30. Method summary

rank(String, OOV) Method to ranking the candidates.

8.6.31. Constructors

- **CandidatesRanking**

```
public CandidatesRanking()
```

  - **Description**
    Constructor of the class.

8.6.32. Methods

- **rank**

```
public java.util.List rank(java.lang.String text,com.
    jmorenov.tweetsccore.extra.OOV oov)
```

  - **Description**
    Method to ranking the candidates.
  - **Parameters**
    - `text` – Original text
    - `oov` – OOV
  - **Returns** – Order List of Candidate

8.6.33. Class FastTextCandidatesMethod

FastTextCandidatesMethod class that define a method to generate candidates.

8.6.34. Declaration

```
public class FastTextCandidatesMethod
 extends com.jmorenov.tweetsccore.candidates.
    CandidatesMethod
```

8.6.35. Constructor summary

FastTextCandidatesMethod() Constructor of the class.

8.6.36. Method summary

generateCandidates(OOV) Method to generate condidates from an OOV.
getMethod() Method to obtain the method description.

8.6.37. Constructors

- **FastTextCandidatesMethod**

  ```
  public FastTextCandidatesMethod()
  ```

  - **Description**
    Constructor of the class.

8.6.38. Methods

- **generateCandidates**

  ```
  public java.util.List generateCandidates(com.jmorenov.
     tweetsccore.extra.OOV oov)
  ```

  - **Description**
    Method to generate condidates from an OOV.
  - **Parameters**
    - `oov` – OOV
  - **Returns** – List of Candidates

- **getMethod**

```
public CandidatesMethodType getMethod()
```

- **Description**
  Method to obtain the method description.
- **Returns** – CandidatesMethodType

8.6.39. Members inherited from class CandidatesMethod

`com.jmorenov.tweetsccore.candidates.CandidatesMethod` (in 8.6.13, page 58)

- `public abstract List` **generateCandidates**`(com.jmorenov.tweetsccore.extra.OOV oov)`
- `public abstract CandidatesMethodType` **getMethod**`()`

8.6.40. Class LevenshteinFSTCandidatesMethod

LevenshteinFSTCandidatesMethod class that define a method to generate candidates.

8.6.41. Declaration

```
public class LevenshteinFSTCandidatesMethod
 extends com.jmorenov.tweetsccore.candidates.
    CandidatesMethod
```

8.6.42. Constructor summary

LevenshteinFSTCandidatesMethod() Constructor of the class.

8.6.43. Method summary

generateCandidates(OOV) Method to generate condidates from an OOV.
getMethod() Method to obtain the method description.

8.6.44. Constructors

- **LevenshteinFSTCandidatesMethod**

```
public LevenshteinFSTCandidatesMethod()
```

- **Description**
  Constructor of the class.

8.6.45. Methods

- **generateCandidates**

```
public java.util.List generateCandidates(com.jmorenov.
    tweetsccore.extra.OOV oov)
```

  - **Description**
    Method to generate condidates from an OOV.
  - **Parameters**
    - `oov` – OOV
  - **Returns** – List of Candidates

- **getMethod**

```
public CandidatesMethodType getMethod()
```

  - **Description**
    Method to obtain the method description.
  - **Returns** – CandidatesMethodType

8.6.46. Members inherited from class CandidatesMethod

`com.jmorenov.tweetsccore.candidates.CandidatesMethod` (in 8.6.13, page 58)

- `public abstract List` **`generateCandidates`**`(com.jmorenov.tweetsccore.extra.OOV` **`oov`**`)`
- `public abstract CandidatesMethodType` **`getMethod`**`()`

8.6.47. Class MetaphoneCandidatesMethod

MetaphoneCandidatesMethod class that define a method to generate candidates.

8.6.48. Declaration

```
public class MetaphoneCandidatesMethod
 extends com.jmorenov.tweetsccore.candidates.
    CandidatesMethod
```

8.6.49. Constructor summary

MetaphoneCandidatesMethod() Constructor of the class.

8.6.50. Method summary

generateCandidates(OOV) Method to generate condidates from an OOV.
getMethod() Method to obtain the method description.

8.6.51. Constructors

- **MetaphoneCandidatesMethod**

```
public MetaphoneCandidatesMethod()
```

  - **Description**
    Constructor of the class.

8.6.52. Methods

- **generateCandidates**

```
public java.util.List generateCandidates(com.jmorenov.
    tweetsccore.extra.OOV oov)
```

  - **Description**
    Method to generate condidates from an OOV.
  - **Parameters**
    - `oov` – OOV
  - **Returns** – List of Candidates

- **getMethod**

```
public CandidatesMethodType getMethod()
```

  - **Description**
    Method to obtain the method description.
  - **Returns** – CandidatesMethodType

8.6.53. Members inherited from class CandidatesMethod

`com.jmorenov.tweetsccore.candidates.CandidatesMethod` (in 8.6.13, page 58)

- `public abstract List` **`generateCandidates`**`(com.jmorenov.tweetsccore.extra.OOV` **`oov`**`)`
- `public abstract CandidatesMethodType` **`getMethod`**`()`

8.7. Package com.jmorenov.tweetsccore.evaluation

Package Contents *Page*

Classes

8.7.1. Class TweetNormEvaluationResult

8.7.2. Declaration

```
public class TweetNormEvaluationResult
 extends java.lang.Object
```

8.7.3. Constructor summary

TweetNormEvaluationResult(String)

8.7.4. Method summary

getAccurancy()
getErrors()
getNegatives()
getPositives()
getResultText()

8.7.5. Constructors

- **TweetNormEvaluationResult**

```
public TweetNormEvaluationResult(java.lang.String
    resultText)
```

8.7.6. Methods

- **getAccurancy**

```
public float getAccurancy()
```

- **getErrors**

```
public int getErrors()
```

- **getNegatives**

```
public int getNegatives()
```

- **getPositives**

```
public int getPositives()
```

- **getResultText**

```
public java.lang.String getResultText()
```

8.7.7. Class TweetNormEvaluator

TweetNormEvaluator class to evaluate methods of spell checker with Tweet Norm 2013 files to test.

8.7.8. Declaration

```
public class TweetNormEvaluator
 extends java.lang.Object
```

8.7.9. Constructor summary

TweetNormEvaluator() Default constructor of the class.
TweetNormEvaluator(String) Constructor of the class with parameter.
TweetNormEvaluator(String, boolean) Constructor of the class with parameters.

8.7.10. Method summary

evalutate(Method) Method to evaluate the defined file with a method of spell checker.
setAnnotatedFile(String) Method to define the file with the annotated tweets.
setEvaluatorScriptFile(String) Method to define the file of the evaluator script.
setIdsFile(String) Method to define the file with the ids of the tweets.
setResultFile(String) Method to define the result file.
setTweetsFile(String) Method to define the file with the tweets.
setWorkingDirectory(String) Method to define the working directory.

8.7.11. Constructors

- **TweetNormEvaluator**

```
public TweetNormEvaluator()
```

  - **Description**
    Default constructor of the class.

- **TweetNormEvaluator**

```
public TweetNormEvaluator(java.lang.String annotatedFile
    )
```

  - **Description**
    Constructor of the class with parameter.
  - **Parameters**
    - `annotatedFile` – String parameter with the name of the file with the annotated tweets.

- **TweetNormEvaluator**

```
public TweetNormEvaluator(java.lang.String annotatedFile
    ,boolean verbose)
```

  - **Description**
    Constructor of the class with parameters.
  - **Parameters**
    - `annotatedFile` – String parameter with the name of the file with the annotated tweets.
    - `verbose` – Boolean parameter to define the verbose control.

8.7.12. Methods

- **evalutate**

```
public TweetNormEvaluationResult evalutate(com.jmorenov.
    tweetsccore.method.Method method) throws java.io.
    IOException
```

- **Description**
  Method to evaluate the defined file with a method of spell checker.
- **Parameters**
  - `method` – parameter with the method to use.
- **Returns** – String with the output of the evaluation.
- **Throws**
  - `java.io.IOException` – when the file not found.
- **See also**
  - `com.jmorenov.tweetsccore.method.Method` (in 8.9.8, page 85)

- **setAnnotatedFile**

```
public void setAnnotatedFile(java.lang.String
    annotatedFile)
```

  - **Description**
    Method to define the file with the annotated tweets.
  - **Parameters**
    - `annotatedFile` – String parameter with the name of the file with the annotated tweets.

- **setEvaluatorScriptFile**

```
public void setEvaluatorScriptFile(java.lang.String
    evaluatorScriptFile)
```

  - **Description**
    Method to define the file of the evaluator script.
  - **Parameters**
    - `evaluatorScriptFile` – String parameter with the name of the evaluator script.

- **setIdsFile**

```
public void setIdsFile(java.lang.String idsFile)
```

  - **Description**
    Method to define the file with the ids of the tweets.
  - **Parameters**

- `idsFile` – String parameter with the name of the file with the ids of the tweets.

- **setResultFile**

```
public void setResultFile(java.lang.String resultFile)
```

  - **Description**
    Method to define the result file.
  - **Parameters**
    - `resultFile` – String parameter with the result file.

- **setTweetsFile**

```
public void setTweetsFile(java.lang.String tweetsFile)
```

  - **Description**
    Method to define the file with the tweets.
  - **Parameters**
    - `tweetsFile` – String parameter with the name of the file with the tweets.

- **setWorkingDirectory**

```
public void setWorkingDirectory(java.lang.String
    workingDirectory)
```

  - **Description**
    Method to define the working directory.
  - **Parameters**
    - `workingDirectory` – String parameter with the working directory.

8.8. Package com.jmorenov.tweetsccore.extra

Package Contents *Page*

Classes

8.8.1. Class Annotation

Anotation enum with the different anotations posibilities of a tweet.

8.8.2. Declaration

```
public final class Annotation
 extends java.lang.Enum
```

8.8.3. Field summary

Correct
NoEs
Unknown
value
Variation

8.8.4. Method summary

valueOf(String)
values()

8.8.5. Fields

- `public static final Annotation` **Variation**
- `public static final Annotation` **Correct**
- `public static final Annotation` **NoEs**
- `public static final Annotation` **Unknown**
- `public int` **value**

8.8.6. Methods

- **valueOf**

```
public static Annotation valueOf(java.lang.String name)
```

- **values**

```
public static Annotation[] values()
```

8.8.7. Members inherited from class Enum

`java.lang.Enum`

- `protected final Object` **clone**`() throws CloneNotSupportedException`
- `public final int` **compareTo**`(Enum` **arg0**`)`
- `public final boolean` **equals**`(Object` **arg0**`)`
- `protected final void` **finalize**`()`
- `public final Class` **getDeclaringClass**`()`
- `public final int` **hashCode**`()`
- `public final String` **name**`()`
- `public final int` **ordinal**`()`
- `public String` **toString**`()`
- `public static Enum` **valueOf**`(Class` **arg0**`, String` **arg1**`)`

8.8.8. Class File

File class with funcionality to files.

8.8.9. Declaration

```
public class File
 extends java.lang.Object
```

8.8.10. Constructor summary

File()

8.8.11. Method summary

getStreamFromResources(String) Method to read a file stream from resources.
readDictionaryFromResources(String) Method ro read a dictionary from resources.
readToByte(String) Method to read a file to byte from resources.
readToStringArray(String) Method to read a file to array of string.

8.8.12. Constructors

- **File**

```
public File()
```

8.8.13. Methods

- **getStreamFromResources**

```
public static java.io.InputStream getStreamFromResources
    (java.lang.String fileName) throws java.io.
    IOException
```

  - **Description**
    Method to read a file stream from resources.
  - **Parameters**
    - `fileName` – the name of the file.
  - **Returns** – InputStream of the file.
  - **Throws**
    - `java.io.IOException` – when the file is not found.

- **readDictionaryFromResources**

```
public static java.util.List readDictionaryFromResources
    (java.lang.String fileName) throws java.io.
    IOException
```

  - **Description**
    Method ro read a dictionary from resources.
  - **Parameters**
    - `fileName` – String
  - **Returns** – List of String
  - **Throws**
    - `java.io.IOException` – When the file is not found

- **readToByte**

```
public static byte[] readToByte(java.lang.String
    fileName) throws java.io.IOException
```

- **Description**
  Method to read a file to byte from resources.
- **Parameters**
  - `fileName` – the name of the file.
- **Returns** – byte[] of the file.
- **Throws**
  - `java.io.IOException` – when the file is not found.

- **readToStringArray**

```
public static java.lang.String[] readToStringArray(java.
    lang.String fileName) throws java.io.IOException
```

  - **Description**
    Method to read a file to array of string.
  - **Parameters**
    - `fileName` – String with the name of the file.
  - **Returns** – String[] with the lines.
  - **Throws**
    - `java.io.IOException` – when the file is not found.

8.8.14. Class FreelingInitializator

FreelingTokenizer class to initialize Freeling.

8.8.15. Declaration

```
public class FreelingInitializator
 extends java.lang.Object
```

8.8.16. Constructor summary

FreelingInitializator()

8.8.17. Method summary

init() Constructor of the class

8.8.18. Constructors

- **FreelingInitializator**

```
public FreelingInitializator()
```

8.8.19. Methods

- **init**

```
public static java.lang.String init()
```

  - **Description**
    Constructor of the class

8.8.20. Class OOV

Out-Of-Vocabulary class with the structure of a OOV word.

8.8.21. Declaration

```
public class OOV
 extends java.lang.Object
```

8.8.22. Constructor summary

OOV(String, int, int) Constructor of the class.
OOV(Token) Constructor of the class.

8.8.23. Method summary

getAnnotation() Method to get the annotation.
getCandidates() Method to get the candidates of an OOV.
getCorrection() Method to get the correction.
getEndPosition() Method to get the end position into the original text.
getStartPosition() Method to get the start position into the original text.
getToken() Method to get the token.
setAnnotation(Annotation) Method to set the annotation
setCandidates(List) Method to set the candidates of an OOV.
setCorrection(String) Method to set the correction.

8.8.24. Constructors

- **OOV**

```
public OOV(java.lang.String token,int startPosition,int
    endPosition)
```

  - **Description**
    Constructor of the class.

- **Parameters**
  - `token` – String with the word or token of the OOV.
  - `startPosition` – int with the initial position of the OOV in the original text.
  - `endPosition` – int with the final position of the OOV in the original text.

- **OOV**

```
public OOV(Token token)
```

  - **Description**
    Constructor of the class.
  - **Parameters**
    - `token` – Token

8.8.25. Methods

- **getAnnotation**

```
public Annotation getAnnotation()
```

  - **Description**
    Method to get the annotation.
  - **Returns** – Annotation

- **getCandidates**

```
public java.util.List getCandidates()
```

  - **Description**
    Method to get the candidates of an OOV.
  - **Returns** – List of Candidate

- **getCorrection**

```
public java.lang.String getCorrection()
```

  - **Description**
    Method to get the correction.

- **Returns** – String

- **getEndPosition**

```
public int getEndPosition()
```

  - **Description**
    Method to get the end position into the original text.
  - **Returns** – int

- **getStartPosition**

```
public int getStartPosition()
```

  - **Description**
    Method to get the start position into the original text.
  - **Returns** – int

- **getToken**

```
public java.lang.String getToken()
```

  - **Description**
    Method to get the token.
  - **Returns** – String

- **setAnnotation**

```
public void setAnnotation(Annotation annotation)
```

  - **Description**
    Method to set the annotation
  - **Parameters**
    - `annotation` – Annotation

- **setCandidates**

```
public void setCandidates(java.util.List candidates)
```

- **Description**
  Method to set the candidates of an OOV.
- **Parameters**
  - `candidates` – List of Candidate

- **setCorrection**

```
public void setCorrection(java.lang.String correction)
```

  - **Description**
    Method to set the correction.
  - **Parameters**
    - `correction` – String

8.8.26. Class OOVDetector

OOVDetector class that define a method to detect OOV.

8.8.27. Declaration

```
public class OOVDetector
 extends java.lang.Object
```

8.8.28. Constructor summary

OOVDetector() Constructor of the class.

8.8.29. Method summary

detectOOV(List) Method to detect the OOV.

8.8.30. Constructors

- **OOVDetector**

```
public OOVDetector()
```

  - **Description**
    Constructor of the class.

8.8.31. Methods

- **detectOOV**

```
public java.util.List detectOOV(java.util.List tokens)
```

  - **Description**
    Method to detect the OOV.
  - **Parameters**
    - `tokens` – List of Token
  - **Returns** – List of OOV

8.8.32. Class Parser

8.8.33. Declaration

```
public class Parser
 extends java.lang.Object
```

8.8.34. Constructor summary

Parser()

8.8.35. Method summary

getHashtagRegex() Method to get the hashtag regex pattern.
getURLRegex() Method to get the url regex pattern.
getUsernameRegex() Method to get the user name regex pattern.
isHashtag(String) Method to check if a word is a hashtag of Twitter.
isPunctuationSign(String) Method to check if a word is a punctuation sign.
isUrl(String) Method to check if a word is a Url.
isUsername(String) Method to check if a word is a username of Twitter.
isValidWord(String) Method to check if a word is a valid word.
removeEmojiFromText(String) Method to remove the emojis from a text.

8.8.36. Constructors

- **Parser**

```
public Parser()
```

8.8.37. Methods

- **getHashtagRegex**

  ```
  public static java.lang.String getHashtagRegex()
  ```

  - **Description**
    Method to get the hashtag regex pattern.
  - **Returns** – String with the pattern

- **getURLRegex**

  ```
  public static java.lang.String getURLRegex()
  ```

  - **Description**
    Method to get the url regex pattern.
  - **Returns** – String with the pattern

- **getUsernameRegex**

  ```
  public static java.lang.String getUsernameRegex()
  ```

  - **Description**
    Method to get the user name regex pattern.
  - **Returns** – String with the pattern

- **isHashtag**

  ```
  public static java.lang.Boolean isHashtag(java.lang.
     String word)
  ```

  - **Description**
    Method to check if a word is a hashtag of Twitter.
  - **Parameters**
    - `word` – String with the word to check.
  - **Returns** – Boolean control parameter.

- **isPunctuationSign**

```
public static java.lang.Boolean isPunctuationSign(java.
    lang.String word)
```

- **Description**
  Method to check if a word is a punctuation sign.
- **Parameters**
  - `word` – String with the word to check.
- **Returns** – Boolean control parameter.

- **isUrl**

```
public static java.lang.Boolean isUrl(java.lang.String
    word)
```

- **Description**
  Method to check if a word is a Url.
- **Parameters**
  - `word` – String with the word to check.
- **Returns** – Boolean control parameter.

- **isUsername**

```
public static java.lang.Boolean isUsername(java.lang.
    String word)
```

- **Description**
  Method to check if a word is a username of Twitter.
- **Parameters**
  - `word` – String with the word to check.
- **Returns** – Boolean control parameter.

- **isValidWord**

```
public static java.lang.Boolean isValidWord(java.lang.
    String word)
```

- **Description**
  Method to check if a word is a valid word.
- **Parameters**

- `word` – String with the word to check.

- **Returns** – Boolean control parameter.

- **removeEmojiFromText**

```
public static java.lang.String removeEmojiFromText(java.
    lang.String text)
```

- **Description**
  Method to remove the emojis from a text.
- **Parameters**
  - `text` – String with the text to remove the emojis.
- **Returns** – String with the text without the emojis.

8.8.38. Class Token

Token class with the structure of a token from a text.

8.8.39. Declaration

```
public class Token
 extends java.lang.Object
```

8.8.40. Constructor summary

Token(String, int, int) Constructor of the class.

8.8.41. Method summary

getEndPosition() Method to get the end position
getStartPosition() Method to get the start position.
getText() Method to get the text of the token.

8.8.42. Constructors

- **Token**

```
public Token(java.lang.String text,int startPosition,int
    endPosition)
```

- **Description**
  Constructor of the class.
- **Parameters**

- `text` – String the text of the token
- `startPosition` – int the start position of the token in the text
- `endPosition` – int the end position of the token in the text

8.8.43. Methods

- **getEndPosition**

```
public int getEndPosition()
```

  - **Description**
    Method to get the end position
  - **Returns** – int

- **getStartPosition**

```
public int getStartPosition()
```

  - **Description**
    Method to get the start position.
  - **Returns** – int

- **getText**

```
public java.lang.String getText()
```

  - **Description**
    Method to get the text of the token.
  - **Returns** – String

8.9. Package com.jmorenov.tweetsccore.method

Package Contents *Page*

Classes

8.9.1. Class DictionaryMethod

DictionaryMethod class with the method of spell checker with dictionaries.

8.9.2. Declaration

```
public class DictionaryMethod
 extends com.jmorenov.tweetsccore.method.Method
```

8.9.3. Constructor summary

DictionaryMethod() Default constructor of the class.

8.9.4. Method summary

correctTweet(Tweet) Method to get the corrected tweet.
toString() Method to get the String of the method.

8.9.5. Constructors

- **DictionaryMethod**

```
public DictionaryMethod() throws java.io.IOException
```

  - **Description**
    Default constructor of the class.
  - **Throws**
    - `java.io.IOException` – when the file not found.

8.9.6. Methods

- **correctTweet**

```
public com.jmorenov.tweetsccore.twitter.TweetCorrected
    correctTweet(com.jmorenov.tweetsccore.twitter.Tweet
    tweet)
```

  - **Description**
    Method to get the corrected tweet.
  - **Parameters**
    - `tweet` – Tweet with the tweet to correct.
  - **Returns** – TweetCorrected with the corrected tweet.

- **toString**

```
public java.lang.String toString()
```

  - **Description**
    Method to get the String of the method.
  - **Returns** – String with the String of the method.

8.9.7. Members inherited from class Method

`com.jmorenov.tweetsccore.method.Method` (in 8.9.8, page 85)

- `public abstract TweetCorrected` **correctTweet**`(com.jmorenov.tweetsccore.twitter.Tw`
  tweet)
- `public abstract String` **toString**`()`

8.9.8. Class Method

Method abstract class.

8.9.9. Declaration

```
public abstract class Method
 extends java.lang.Object
```

8.9.10. All known subclasses

TweetSCMethod (in 8.9.15, page 86), DictionaryMethod (in 8.9.1, page 84)

8.9.11. Constructor summary

Method() Default constructor of the class.

8.9.12. Method summary

correctTweet(Tweet) Abstract method to get the corrected tweet.
toString() Abstract method to get the String of the method.

8.9.13. Constructors

- **Method**

```
public Method()
```

  - **Description**
    Default constructor of the class.

8.9.14. Methods

- **correctTweet**

```
public abstract com.jmorenov.tweetsccore.twitter.
    TweetCorrected correctTweet(com.jmorenov.tweetsccore.
    twitter.Tweet tweet)
```

  - **Description**
    Abstract method to get the corrected tweet.
  - **Parameters**
    - `tweet` – Tweet with the tweet to correct.
  - **Returns** – TweetCorrected with the corrected tweet.

- **toString**

```
public abstract java.lang.String toString()
```

  - **Description**
    Abstract method to get the String of the method.
  - **Returns** – String with the String of the method.

8.9.15. Class TweetSCMethod

TweetSCMethod class that define a method to spell check.

8.9.16. Declaration

```
public class TweetSCMethod
 extends com.jmorenov.tweetsccore.method.Method
```

8.9.17. Constructor summary

TweetSCMethod() Constructor of the class.

8.9.18. Method summary

correctTweet(Tweet) Method to correct a tweet.
toString() Method to get the description of the method.

8.9.19. Constructors

- **TweetSCMethod**

```
public TweetSCMethod()
```

  - **Description**
    Constructor of the class.

8.9.20. Methods

- **correctTweet**

```
public com.jmorenov.tweetsccore.twitter.TweetCorrected
    correctTweet(com.jmorenov.tweetsccore.twitter.Tweet
    tweet)
```

  - **Description**
    Method to correct a tweet.
  - **Parameters**
    - `tweet` – Tweet with the tweet to correct.
  - **Returns** – TweetCorrected

- **toString**

```
public java.lang.String toString()
```

  - **Description**
    Method to get the description of the method.
  - **Returns** – String

8.9.21. Members inherited from class Method

`com.jmorenov.tweetsccore.method.Method` (in 8.9.8, page 85)

- `public abstract TweetCorrected` **correctTweet**`(com.jmorenov.tweetsccore.twitter.Tw`
  tweet)
- `public abstract String` **toString**`()`

8.10. Package com.jmorenov.tweetsccore.post

Package Contents *Page*

Classes

8.10.1. Class FreeLingPOST

FreeLingPOST class to get the POST of a text. https://talp-upc.gitbooks.io/freeling-4-1-user-manual/content/

8.10.2. Declaration

```
public class FreeLingPOST
 extends java.lang.Object
```

8.10.3. Constructor summary

FreeLingPOST()

8.10.4. Constructors

- **FreeLingPOST**

```
public FreeLingPOST()
```

8.10.5. Class OpenNLPPOST

OpenNLPPOST class to get the POST of a text. https://opennlp.apache.org/docs/1.8.4/manual/openn

8.10.6. Declaration

```
public class OpenNLPPOST
 extends java.lang.Object
```

8.10.7. Constructor summary

OpenNLPPOST()

8.10.8. Constructors

- **OpenNLPPOST**

```
public OpenNLPPOST()
```

8.10.9. Class POST

8.10.10. Declaration

```
public abstract class POST
 extends java.lang.Object
```

8.10.11. All known subclasses

StanfordNLPPOST (in 8.10.16, page 89)

8.10.12. Constructor summary

POST()

8.10.13. Method summary

getTags()

8.10.14. Constructors

- **POST**

```
public POST()
```

8.10.15. Methods

- **getTags**

```
public abstract java.lang.String getTags()
```

8.10.16. Class StanfordNLPPOST

8.10.17. Declaration

```
public class StanfordNLPPOST
 extends com.jmorenov.tweetsccore.post.POST
```

8.10.18. Constructor summary

StanfordNLPPOST(String)

8.10.19. Method summary

getTags()

8.10.20. Constructors

- **StanfordNLPPOST**

```
public StanfordNLPPOST(java.lang.String text)
```

8.10.21. Methods

- **getTags**

```
public abstract java.lang.String getTags()
```

8.10.22. Members inherited from class POST

`com.jmorenov.tweetsccore.post.POST` (in 8.10.9, page 89)

- `public abstract String getTags()`

8.11. Package com.jmorenov.tweetsccore.spellchecker

Package Contents *Page*

Classes

SpellChecker 90
SpellChecker class to correct a text.

8.11.1. Class SpellChecker

SpellChecker class to correct a text.

8.11.2. Declaration

```
public class SpellChecker
 extends java.lang.Object
```

8.11.3. Constructor summary

SpellChecker(Method) Constructor of the class.

8.11.4. Method summary

correctText(String) Method to correct the text.
correctTweet(Tweet) Method to correct a tweet.
getMethodDescription() Method to get spell checker method description.
setMethod(Method) Method to define the spell checker method.

8.11.5. Constructors

- **SpellChecker**

```
public SpellChecker(com.jmorenov.tweetsccore.method.
    Method method)
```

  - **Description**
    Constructor of the class.
  - **Parameters**
    - `method` – parameter with the method to use.
  - **See also**
    - `com.jmorenov.tweetsccore.method.Method` (in 8.9.8, page 85)

8.11.6. Methods

- **correctText**

```
public java.lang.String correctText(java.lang.String
    text)
```

  - **Description**
    Method to correct the text.
  - **Parameters**
    - `text` – String with the text to correct.
  - **Returns** – String with the corrected text.

- **correctTweet**

```
public com.jmorenov.tweetsccore.twitter.TweetCorrected
    correctTweet(com.jmorenov.tweetsccore.twitter.Tweet
    tweet)
```

- **Description**
  Method to correct a tweet.
- **Parameters**
  - `tweet` – Tweet with the tweet.
- **Returns** – TweetCorrected with the corrected tweet.

▪ **getMethodDescription**

```
public java.lang.String getMethodDescription()
```

- **Description**
  Method to get spell checker method description.
- **Returns** – String with the description of the method.

▪ **setMethod**

```
public void setMethod(com.jmorenov.tweetsccore.method.
    Method method)
```

- **Description**
  Method to define the spell checker method.
- **Parameters**
  - `method` – parameter with the method to use.

8.12. Package com.jmorenov.tweetsccore.tokenizer

Package Contents *Page*

Classes

8.12.1. Class FreelingTokenizer

FreelingTokenizer class to tokenize a text.

8.12.2. Declaration

```
public class FreelingTokenizer
 extends com.jmorenov.tweetsccore.tokenizer.Tokenizer
```

8.12.3. Constructor summary

FreelingTokenizer() Constructor of the class

8.12.4. Method summary

getTokens(String) Method to get the tokens from the text.

8.12.5. Constructors

- **FreelingTokenizer**

  ```
  public FreelingTokenizer()
  ```

  - **Description**
    Constructor of the class

8.12.6. Methods

- **getTokens**

  ```
  public java.util.List getTokens(java.lang.String text)
  ```

  - **Description**
    Method to get the tokens from the text.
  - **Parameters**
    - `text` – String with the text
  - **Returns** – List of String with the tokens

8.12.7. Members inherited from class Tokenizer

`com.jmorenov.tweetsccore.tokenizer.Tokenizer` (in 8.12.22, page 95)

- `public abstract List` **`getTokens`**`(java.lang.String` **`text`**`)`

8.12.8. Class OpenNLPTokenizer

OpenNLPTokenizer class to tokenize a text. https://opennlp.apache.org/docs/1.8.4/manual/

8.12.9. Declaration

```
public class OpenNLPTokenizer
 extends com.jmorenov.tweetsccore.tokenizer.Tokenizer
```

8.12.10. Constructor summary

OpenNLPTokenizer() Constructor of the class

8.12.11. Method summary

getTokens(String) Method to get the tokens from the text.

8.12.12. Constructors

- **OpenNLPTokenizer**

```
public OpenNLPTokenizer() throws java.io.IOException
```

  - **Description**
    Constructor of the class
  - **Throws**
    - `java.io.IOException` –

8.12.13. Methods

- **getTokens**

```
public java.util.List getTokens(java.lang.String text)
```

  - **Description**
    Method to get the tokens from the text.
  - **Parameters**
    - `text` – String with the text
  - **Returns** – List of String with the tokens

8.12.14. Members inherited from class Tokenizer

`com.jmorenov.tweetsccore.tokenizer.Tokenizer` (in 8.12.22, page 95)

- `public abstract List` **`getTokens`**`(java.lang.String text)`

8.12.15. Class StanfordNLPTokenizer

StanfordNLPTokenizer class to tokenize a text. https://stanfordnlp.github.io/CoreNLP/

8.12.16. Declaration

```
public class StanfordNLPTokenizer
 extends com.jmorenov.tweetsccore.tokenizer.Tokenizer
```

8.12.17. Constructor summary

StanfordNLPTokenizer() Constructor of the class

8.12.18. Method summary

getTokens(String) Method to get the tokens from the text.

8.12.19. Constructors

- **StanfordNLPTokenizer**

```
public StanfordNLPTokenizer()
```

  - **Description**
    Constructor of the class

8.12.20. Methods

- **getTokens**

```
public java.util.List getTokens(java.lang.String text)
```

  - **Description**
    Method to get the tokens from the text.
  - **Parameters**
    - `text` – String with the text
  - **Returns** – List of String with the tokens

8.12.21. Members inherited from class Tokenizer

`com.jmorenov.tweetsccore.tokenizer.Tokenizer` (in 8.12.22, page 95)

- `public abstract List` **getTokens**`(java.lang.String` **text**`)`

8.12.22. Class Tokenizer

Tokenizer abstract class.

8.12.23. Declaration

```
public abstract class Tokenizer
 extends java.lang.Object
```

8.12.24. All known subclasses

FreelingTokenizer (in 8.12.1, page 92), StanfordNLPTokenizer (in 8.12.15, page 94), OpenNLPTokenizer (in 8.12.8, page 93)

8.12.25. Constructor summary

Tokenizer()

8.12.26. Method summary

getTokens(String) Method to get the tokens from a text.

8.12.27. Constructors

- **Tokenizer**

```
public Tokenizer()
```

8.12.28. Methods

- **getTokens**

```
public abstract java.util.List getTokens(java.lang.
    String text)
```

  - **Description**
    Method to get the tokens from a text.
  - **Parameters**
    - `text` – String with the text
  - **Returns** – List of Token

9. TweetSCWeb Documentación del código (Java Documentation)

Class Hierarchy

Classes

- java.lang.Object
  - SpringBootServletInitializer
    - com.jmorenov.tweetscweb.ServletInitializer (in 9.1.13, page 100)
  - com.jmorenov.tweetscweb.Application (in 9.1.1, page 98)
  - com.jmorenov.tweetscweb.Response (in 9.1.7, page 98)
  - com.jmorenov.tweetscweb.TweetCorrectedListModel (in 9.1.19, page 101)
  - com.jmorenov.tweetscweb.TweetCorrectorAPIController (in 9.1.32, page 102)
  - com.jmorenov.tweetscweb.TweetCorrectorController (in 9.1.38, page 104)
  - com.jmorenov.tweetscweb.TweetListModel (in 9.1.44, page 105)
  - com.jmorenov.tweetscweb.TweetModel (in 9.1.50, page 106)
    - com.jmorenov.tweetscweb.TweetCorrectedModel (in 9.1.25, page 102)
  - com.jmorenov.tweetscweb.TweetSearchQuery (in 9.1.59, page 107)
  - com.jmorenov.tweetscweb.TweetSearchQueryModel (in 9.1.65, page 108)

9.1. Package com.jmorenov.tweetscweb

Package Contents *Page*

Classes

TweetSearchQuery class with the functionality to work over the queries.

TweetSearchQueryModel 108

TweetSearchQueryModel class with the model of the queries.

9.1.1. Class Application

Application class.

9.1.2. Declaration

```
public class Application
 extends java.lang.Object
```

9.1.3. Constructor summary

Application()

9.1.4. Method summary

main(String[]) Main method of the web application.

9.1.5. Constructors

- **Application**

```
public Application()
```

9.1.6. Methods

- **main**

```
public static void main(java.lang.String[] args)
```

  - **Description**
    Main method of the web application.
  - **Parameters**
    - `args` – String[] with the arguments of the execution.

9.1.7. Class Response

Response class.

9.1.8. Declaration

```
public class Response
 extends java.lang.Object
```

9.1.9. Constructor summary

Response() Default constructor of the class.
Response(String, Object) Constructor of the class.

9.1.10. Method summary

getData() Method to get the data of the response
getStatus() Method to get the status of the response.
setData(Object) Method to define the data of the response
setStatus(String) Method to define the status of the response.

9.1.11. Constructors

- **Response**

```
public Response()
```

  - **Description**
    Default constructor of the class.

- **Response**

```
public Response(java.lang.String status,java.lang.Object
    data)
```

  - **Description**
    Constructor of the class.
  - **Parameters**
    - `status` – String with the status of the response.
    - `data` – Object with the value of the response.

9.1.12. Methods

- **getData**

```
public java.lang.Object getData()
```

- **Description**
  Method to get the data of the response
- **Returns** – Object with the value of the response.

- **getStatus**

```
public java.lang.String getStatus()
```

  - **Description**
    Method to get the status of the response.
  - **Returns** – String with the status of the response.

- **setData**

```
public void setData(java.lang.Object data)
```

  - **Description**
    Method to define the data of the response
  - **Parameters**
    - `data` – Object with the value of the response.

- **setStatus**

```
public void setStatus(java.lang.String status)
```

  - **Description**
    Method to define the status of the response.
  - **Parameters**
    - `status` – String with the status.

9.1.13. Class ServletInitializer

ServletInitializer class.

9.1.14. Declaration

```
public class ServletInitializer
 extends SpringBootServletInitializer
```

9.1.15. Constructor summary

ServletInitializer()

9.1.16. Method summary

configure(SpringApplicationBuilder)

9.1.17. Constructors

- **ServletInitializer**

```
public ServletInitializer()
```

9.1.18. Methods

- **configure**

```
protected SpringApplicationBuilder configure(
    SpringApplicationBuilder application)
```

9.1.19. Class TweetCorrectedListModel

TweetCorrectedListModel class with the model of the list of tweets corrected.

9.1.20. Declaration

```
public class TweetCorrectedListModel
 extends java.lang.Object
```

9.1.21. Field summary

tweets

9.1.22. Constructor summary

TweetCorrectedListModel()

9.1.23. Fields

- `public java.util.List` **tweets**

9.1.24. Constructors

- **TweetCorrectedListModel**

```
public TweetCorrectedListModel()
```

9.1.25. Class TweetCorrectedModel

9.1.26. Declaration

```
public class TweetCorrectedModel
 extends com.jmorenov.tweetscweb.TweetModel
```

9.1.27. Field summary

correctedText

9.1.28. Constructor summary

TweetCorrectedModel()
TweetCorrectedModel(TweetCorrected)

9.1.29. Fields

- `public java.lang.String` **correctedText**

9.1.30. Constructors

- **TweetCorrectedModel**

```
public TweetCorrectedModel()
```

- **TweetCorrectedModel**

```
public TweetCorrectedModel(TweetCorrected tweetCorrected
    )
```

9.1.31. Members inherited from class TweetModel

`com.jmorenov.tweetscweb.TweetModel` (in 9.1.50, page 106)

- `public` **date**
- `public` **hash**
- `public` **id**
- `public` **text**
- `public Tweet` **toTweet()**
- `public` **username**

9.1.32. Class TweetCorrectorAPIController

TweetCorrectorAPIController class with the controller of the API.

9.1.33. Declaration

```
public class TweetCorrectorAPIController
 extends java.lang.Object
```

9.1.34. Constructor summary

TweetCorrectorAPIController()

9.1.35. Method summary

advancedCorrectSubmit(TweetListModel) Method to control the api calls of advanced corrector.

getTweets(TweetSearchQueryModel) Method to control the api calls.

simpleCorrectSubmit(TweetModel) Method to control the api calls of simple corrector.

9.1.36. Constructors

- **TweetCorrectorAPIController**

  ```
  public TweetCorrectorAPIController()
  ```

9.1.37. Methods

- **advancedCorrectSubmit**

  ```
  public Response advancedCorrectSubmit(TweetListModel
    tweetListModel)
  ```

  - **Description**
    Method to control the api calls of advanced corrector.
  - **Parameters**
    - `tweetListModel` – with the model of the call.
  - **Returns** – with the response of the call.

- **getTweets**

  ```
  public Response getTweets(TweetSearchQueryModel
    tweetSearchQueryModel)
  ```

  - **Description**
    Method to control the api calls.

- **Parameters**
  - `tweetSearchQueryModel` – with the model of the call.
- **Returns** – with the response of the call.

- **simpleCorrectSubmit**

```
public Response simpleCorrectSubmit(TweetModel
    tweetModel)
```

  - **Description**
    Method to control the api calls of simple corrector.
  - **Parameters**
    - `tweetModel` – with the model of the call.
  - **Returns** – with the response of the call.

9.1.38. Class TweetCorrectorController

TweetCorrectorController class with the controller of the frontend.

9.1.39. Declaration

```
public class TweetCorrectorController
 extends java.lang.Object
```

9.1.40. Constructor summary

TweetCorrectorController()

9.1.41. Method summary

homeForm(Model) Method to control the frontend calls.

9.1.42. Constructors

- **TweetCorrectorController**

```
public TweetCorrectorController()
```

9.1.43. Methods

- **homeForm**

```
public java.lang.String homeForm(Model model)
```

  - **Description**
    Method to control the frontend calls.
  - **Parameters**
    - `model` – Model with the model of the call.
  - **Returns** – String with the template to show.

9.1.44. Class TweetListModel

TweetListModel class with the model of the list of tweets.

9.1.45. Declaration

```
public class TweetListModel
 extends java.lang.Object
```

9.1.46. Field summary

tweets

9.1.47. Constructor summary

TweetListModel()

9.1.48. Fields

- `public java.util.List` **tweets**

9.1.49. Constructors

- **TweetListModel**

```
public TweetListModel()
```

9.1.50. Class TweetModel

9.1.51. Declaration

```
public class TweetModel
 extends java.lang.Object
```

9.1.52. All known subclasses

TweetCorrectedModel (in 9.1.25, page 102)

9.1.53. Field summary

date
hash
id
text
username

9.1.54. Constructor summary

TweetModel()
TweetModel(Tweet)

9.1.55. Method summary

toTweet()

9.1.56. Fields

- `public java.lang.String` **id**
- `public java.lang.String` **username**
- `public java.lang.String` **hash**
- `public java.lang.String` **text**
- `public java.lang.String` **date**

9.1.57. Constructors

- **TweetModel**

  ```
  public TweetModel()
  ```

- **TweetModel**

```
public TweetModel(Tweet tweet)
```

9.1.58. Methods

- **toTweet**

```
public Tweet toTweet()
```

9.1.59. Class TweetSearchQuery

TweetSearchQuery class with the functionality to work over the queries.

9.1.60. Declaration

```
public class TweetSearchQuery
 extends java.lang.Object
```

9.1.61. Constructor summary

TweetSearchQuery(TweetSearchQueryModel) Constructor of the class.

9.1.62. Method summary

isValidQuery() Method to get if the query is valid or not.
loadTweets() Method to load the tweets from the query.

9.1.63. Constructors

- **TweetSearchQuery**

```
public TweetSearchQuery(TweetSearchQueryModel
    tweetSearchQueryModel)
```

- **Description**

  Constructor of the class.

- **Parameters**

  - `tweetSearchQueryModel` – TweetSearchQueryModel with the data.

9.1.64. Methods

- **isValidQuery**

```
public boolean isValidQuery()
```

  - **Description**
    Method to get if the query is valid or not.
  - **Returns** – Boolean

- **loadTweets**

```
public java.util.List loadTweets()
```

  - **Description**
    Method to load the tweets from the query.
  - **Returns** – List of tweet

9.1.65. Class TweetSearchQueryModel

TweetSearchQueryModel class with the model of the queries.

9.1.66. Declaration

```
public class TweetSearchQueryModel
 extends java.lang.Object
```

9.1.67. Constructor summary

TweetSearchQueryModel()

9.1.68. Method summary

getQuery() Method to get the query.
getTweets() Method to get the tweets.
setQuery(String) Method to set the query.
setTweets(List) Method to set the tweets.

9.1.69. Constructors

- **TweetSearchQueryModel**

```
public TweetSearchQueryModel()
```

9.1.70. Methods

- **getQuery**

  ```
  public java.lang.String getQuery()
  ```

  - **Description**
    Method to get the query.
  - **Returns** – String

- **getTweets**

  ```
  public java.util.List getTweets()
  ```

  - **Description**
    Method to get the tweets.
  - **Returns** – List of tweet

- **setQuery**

  ```
  public void setQuery(java.lang.String query)
  ```

  - **Description**
    Method to set the query.
  - **Parameters**
    - `query` – String

- **setTweets**

  ```
  public void setTweets(java.util.List tweets)
  ```

  - **Description**
    Method to set the tweets.
  - **Parameters**
    - `tweets` – List of tweet

10. TweetSCExecutable Documentación del código (Java Documentation)

Class Hierarchy

Classes

- java.lang.Object
  - com.jmorenov.tweetscexecutable.SpellCheckerRun (in 10.1.1, page 111)

10.1. Package com.jmorenov.tweetscexecutable

Package Contents *Page*

Classes

10.1.1. Class SpellCheckerRun

SpellCheckerRun class.

10.1.2. Declaration

```
public class SpellCheckerRun
 extends java.lang.Object
```

10.1.3. Constructor summary

SpellCheckerRun()

10.1.4. Method summary

main(String[]) Main method

10.1.5. Constructors

- **SpellCheckerRun**

```
public SpellCheckerRun()
```

10.1.6. Methods

- **main**

public static void main(java.lang.String[] args) **throws** java.io.IOException

- **Description**
  Main method
- **Parameters**
  - `args` – String[] with the arguments.
- **Throws**
  - `java.io.IOException` – when the files are not found.

11. ANEXOS

11.1. Glosario de términos

- **Modelo (estadístico) del lenguaje:** Un modelo estadístico del lenguaje es una distribución de probabilidad sobre secuencias de palabras. Un tipo de modelo del lenguaje es el unigrama, también se suele llamar modelo de bolsa de palabras. La dispersidad en los datos es un problema al construir modelos del lenguaje. La secuencia de palabras más probable puede no aparecer en los datos de entrenamiento. Una solución es realizar la suposición de que la probabilidad de una palabra sólo depende de las n palabras previas. Esto es conocido como el modelo n-grama, unigrama cuando n=1. Los modelos del lenguaje neuronales o modelos del lenguaje continuos: modelo del lenguaje Skip-gram, base de word2vec.

- **Modelo del lenguaje N-grama:** Un modelo de n-grama es un tipo de modelo probabilístico que permite hacer predicción estadística del próximo elemento de cierta secuencia de elementos sucedida hasta el momento. Un modelo de n-grama puede ser definido por una cadena de Márkov de orden n-1. Predice x_i basándose en los n elementos anteriores.

- **Cadena de Márkov:** En la teoría de la probabilidad, se conoce como cadena de Márkov o modelo de Márkov a un tipo especial de proceso estocástico discreto en el que la probabilidad de que ocurra un evento depende solamente del evento inmediatamente anterior. En matemáticas se define como un proceso estocástico discreto que cumple con la propiedad de Márkov, es decir, si se conoce la historia del sistema hasta su instante actual, su estado presente resume toda la información relevante para describir en probabilidad su futuro.

- **Proceso de Márkov:** Fenómeno aleatorio dependiente del tiempo para el cual se cumple la propiedad de Márkov. Frecuentemente el término cadena de Márkov se usa para dar a entender que un proceso de Márkov tiene un espacio de estados discreto (infinito o numerable).

- **Modelo oculto de Márkov:** Un modelo oculto de Márkov (Hidden Markov Model, HMM) es un modelo estadístico en el que se asume que el sistema a modelar es un proceso de Márkov de parámetros desconocidos. El objetivo es determinar los parámetros desconocidos (u ocultos) de dicha cadena a partir de los parámetros observables. Un HMM se puede considerar como la red bayesiana más simple.

- **Etiquetado gramatical:** El etiquetado gramatical (part-of-speech tagging, POS tagging o POST) se considera el proceso de asignar a cada palabra de un texto su categoría gramatical. Las soluciones se pueden dividir en dos grandes grupos: aproximaciones lingüísticas basadas en un conjunto de reglas establecidas manualmente por expertos aprendidas de forma (semi)automática, y

las aproximaciones de aprendizaje automático que usan textos, generalmente anotados, para establecer los modelos. Además se pueden encontrar aproximaciones híbridas que combinan ciertos aspectos de las anteriores.

Referencias

[1] Alicia Ageno, Pere R. Comas, Lluís Padró, and Jordi Turmo. The talp-upc approach to tweet-norm 2013, 2013.

[2] Iñaki Alegria, Nora Aranberri, Pere R. Comas, Víctor Fresno, Pablo Gamallo, Lluis Padró, Iñaki San Vicente, Jordi Turmo, and Arkaitz Zubiaga. Tweetnorm: a benchmark for lexical normalization of spanish tweets, 2015.

[3] Iñaki Alegria, Nora Aranberri, Víctor Fresno, Pablo Gamallo, Lluis Padró, Iñaki San Vicente, Jordi Turmo, and Arkaitz Zubiaga. Introducción a la tarea compartida tweet-norm 2013: Normalización léxica de tuits en español, 2013.

[4] Apache. Opennlp. http://opennlp.apache.org.

[5] AiTi Aw, Min Zhang, Juan Xiao, and Jian Su. A phrase-based statistical model for sms text normalization, 2009.

[6] Richard Beaufort, Sophie Roekhaut, Louise-Amelie Cougnon, and Cedrick Fairon. A hybrid rule/model-based finite-state framework for normalizing sms messages, 2002.

[7] Y. Bengio, R. Ducharme, and P. Vincent. A neural probabilistic language model, 2003.

[8] Eric Brill and Robert C. Moore. An improved error model for noisy channel spelling correction, 2000.

[9] Jhon Adrián Cerón-Guzmán and Elizabeth León-Guzmán. Lexical normalization of spanish tweets, 2016.

[10] Monojit Choudhury, Rahul Saraf, Vijit Jain, Animesh Mukherjee, Sudeshna Sarkar, and Anupam Basu. Investigation and modeling of the structure of texting language, 2007.

[11] Paul Cook and Suzanne Stevenson. An unsupervised model for text message normalization, 2009.

[12] J.M. Cotelo, F.L. Cruz, J.A. Troyano, and F.J. Ortega. A modular approach for lexical normalization applied to spanish tweets, 2015.

[13] dwyl. english-words. https://github.com/dwyl/english-words.

[14] dylon. liblevenshtein. https://github.com/universal-automata/liblevenshtein-java.

[15] Jacob Eisenstein. What to do about bad language on the internet, 2013.

[16] Facebook. fasttext. https://fasttext.cc/.

[17] Tim Finin, Will Murnane, Anand Karandikar, Nicholas Keller, Justin Martineau, and Mark Dredze. Annotating named entities in twitter data with crowd-sourcing, 2010.

[18] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling, 2005.

[19] Jenny Rose Finkel and Christopher D. Manning. Nested named entity recognition, 2009.

[20] Jennifer Foster, Ozlem Cetinoglu, Joachim Wagner, Joseph Le Roux, Joakim Nivre, Deirdre Hogan, and Josef van Genabith. From news to comment: Resources and benchmarks for parsing the language of web 2.0, 2011.

[21] Freeling. Freeling. http://nlp.lsi.upc.edu/freeling/.

[22] Pablo Gamallo, Marcos García, and Santiago Fernández-Lanza. Word normalization in twitter using finite-state transducers, 2013.

[23] Pablo Gamallo, Marcos García, and José Ramon Pichel. A method to lexical normalisation of tweets, 2013.

[24] Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael, Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. Part-of-speech tagging for twitter: annotation, features, and experiments, 2011.

[25] Jose M. Gomez-Hidalgo, Andrés A. Caurcel-Díaz, and Yovan Iñiguez del Rio. Un método de análisis de lenguaje tipo sms para el castellano, 2013.

[26] Google. Google cloud engine. https://cloud.google.com/.

[27] Stanford NLP Group. Stanfordnlp core. https://stanfordnlp.github.io/CoreNLP/.

[28] Bo Han and Timothy Baldwin. Lexical normalisation of short text messages: Makn sens a twitter, 2011.

[29] Bo han, Paul Cook, and Timothy Baldwin. unimelb: Spanish text normalisation, 2013.

[30] K. Heafield. Faster and smaller language model queries, 2011.

[31] Mans Hulden and Jerid Francom. Weighted and unweighted transducers for tweet normalization, 2013.

[32] Martin Jansche and Steven P. Abney. Information extraction from voicemail transcripts, 2002.

[33] Joseph Kaufmann and Jugal Kalita. Syntactic normalization of twitter messages, 2010.

[34] Catherine Kobus, Franois Yvon, and Graldine Damnati. Transcrire les sms comme on reconnat la parole, 2008.

[35] George R. Krupka and Kevin Hausman. Isoquest: Description of the netowltm extractor system as used in muc-7., 1998.

[36] Xiaohua Liu, Shaodian Zhang, Furu Wei, and Ming Zhou. Recognizing named entities in tweets, 2011.

[37] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Cernocký. Empirical evaluation and combination of advanced language modeling techniques, 2011.

[38] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[39] Einat Minkov, Richard C. Wang, and William W. Cohen. Extracting personal names from email: applying named entity recognition to informal text, 2005.

[40] Javier Moreno. Tweetsc spell checker app. https://jmorenov.github.io/TweetSC/.

[41] Javier Moreno. Tweetsc web. https://tweetsc.github.io.

[42] Mosquera, Alejandro, Elena Lloret, and Paloma Moreda. Towards facilitating the accessibility of web 2.0 texts through text normalisation, 2012.

[43] Alejandro Mosquera. Algoritmo del metáfono. https://github.com/amsqr/Spanish-Metaphone/blob/master/phonetic_algorithms_es.py.

[44] Alejandro Mosquera and Paloma Moreda. Dlsi en tweet-norm 2013: Normalizacion de tweets en español, 2013.

[45] Juan M. Cotelo Moya, Fermín L Cruz, and Jose A. Troyano. Resource-based lexical approach to tweet-norm task, 2013.

[46] Forsyth Eric N. and Craig H. Martell. Lexical and discourse analysis of online chat dialog, 2007.

[47] Peter Norvig. How to write a spelling corrector. http://norvig.com/spell-correct.html, 2007.

[48] Jesús Oliva, José I. Serrano, María D. Del Castillo, and Angel Iglesias. Sms normalization: combining phonetics, morphology and semantics, 2011.

[49] Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. Improved part-of-speech tagging for online conversational text with word clusters, 2013.

[50] James L. Peterson. Computer programs for detecting and correcting spelling errors., 1980.

[51] Bojanowski Piotr, Grave Edouard, Joulin Armand, and Mikolov Tomas. Enriching word vectors with subword information, 2017.

[52] Alan Ritter, Colin Cherry, and Bill Dolan. Unsupervised modeling of twitter conversations, 2010.

[53] Alan Ritter, Sam Clark, Mausam, and Oren Etzioni. Named entity recognition in tweets: an experimental study, 2011.

[54] Pablo Ruiz, Montse Cuadros, and Thierry Etchegoyhen. Lexical normalization of spanish tweets with preprocessing rules, domain-specific edit distances, and language models, 2013.

[55] Pablo Ruiz, Montse Cuadros, and Thierry Etchegoyhen. Lexical normalization of spanish tweets with rule-based components and language models, 2014.

[56] Arturo Montejo Ráez, M. Carlos Diaz Galiano, Eugenio Martíne Cámara, M. Teresa Martín Valdivia, Miguel A. García Cumbreras, and L. Alfonso Ureña López. Sinai at twitter-normalization 2013, 2013.

[57] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: language independent named entity recognition, 2003.

[58] H. Schwenk. Continuous space language models, 2007.

[59] Claude Elwood Shannon. A mathematical theory of communication, 1948.

[60] Beaux Sharifi, Mark-Anthony Hutton, and Jugal Kalita. Summarizing microblogs automatically, 2010.

[61] Sameer Singh, Dustin Hillard, and Chris Leggetter. Minimally-supervised extraction of entities from text advertisements, 2010.

[62] Enrique Sánchez-Villamil, Mikel L. Forcada, and Rafael C. Carrasco. Unsupervised training of a finite-state sliding-window part-of-speech tagger, 2004.

[63] Kristina Toutanova and Robert C. Moore. Pronunciation modeling for improved spelling correction, 2002.

[64] Tweet-Norm. Tweet-norm. http://komunitatea.elhuyar.eus/tweet-norm/.

[65] Xabier Saralegi Urizar and Iñaki San Vicente Roncal. Elhuyar at tweetnorm 2013, 2013.

[66] Jesús Vilares, Miguel A. Alonso, and David Vilares. Prototipado rápido de un sistema de normalización de tuits: Una aproximación léxica, 2013.

[67] Casey Whitelaw, BenHutchinson, Grace Y. Chung, and Gerard Ellis. Using the web for language independent spellchecking and autocorrection, 2009.

[68] GuoDong Zhou and Jian Su. Named entity recognition using an hmm-based chunk tagger., 2002.

[69] Óscar Muñoz-García, Silvia Vázquez, and Nuria Bel. Exploiting web-based collective knowledge for micropost normalisation, 2013.