Análisis de sentimiento

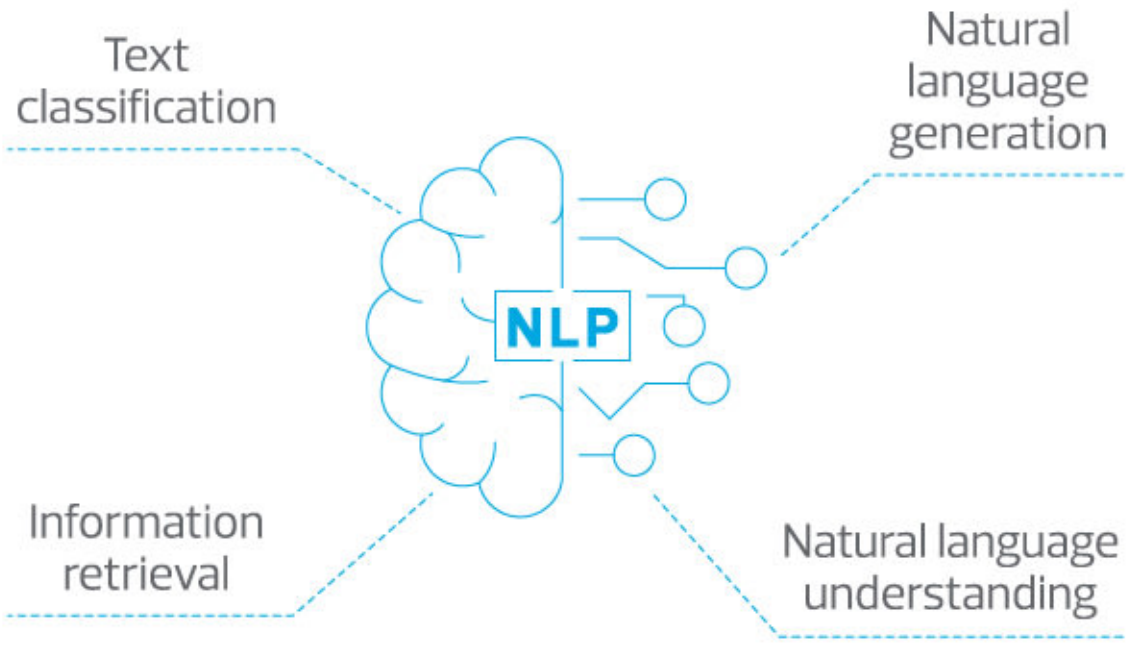
Victor Rodeño Sanchez

8 de noviembre de 2019

Introducción NLP

El procesamiento del lenguaje natural ("natural lenguage proccesing"'en inglés) es el campo que se encarga de entender el lenguaje, su construcción, análisis y las tareas relacionadas con esto que permitan la comunicación con computadoras. Utilizando el machine learning se quiere automatizar este análisis, es decir, que sea la maquina o modelo el que aprenda a hacer este análisis automáticamente y tareas que estén relacionadas con el nlp.

Hay muchas sub tareas relacionadas con el machine learning como traducir texto, chatbots generar texto y otras tareas como captar emociones ...



Motivación

La motivación de este trabajo ha sido aprender y entender las técnicas básicas del procesamiento del lenguaje natural. Se ha utilizado el lenguaje de programación Python.

El conjunto de datos son 100000 críticas de películas divididas en dos datasets de entrenamiento y de test. Las películas representan las filas. Una columna del dataset representa las críticas y otra 'sentiment' representa la variable a predecir, que vale 1 si es una crítica positiva y 0 si es negativa. Es una variable binaria por lo que es una tarea de clasificación de aprendizaje supervisado.

Análisis

Una tarea de NLP se puede dividir en 3 fases:

- Fase de limpieza y transformación de las palabras. En este paso se eliminaría texto que no aporta información como comas, puntos, palabras que no aportan significado (stopwords), posibles modificaciones de las palabras, etc.
- Fase de transformación de las palabras a vectores o vectorización. En este caso existen diversos métodos para esta tarea y veremos desde los más básicos hasta algunos más complejos
- Por último, el entrenamiento y validación de un modelo predictivo o algoritmo utilizando estos datos con el fin de obtener futuras predicciones. En este caso hay una gran variedad de algoritmos y estudiaremos algunos simples y otras un poco más complejos

Estas fases pueden variar en función de la tarea específica que se esté resolviendo. Una idea de los pasos necesarios para una tarea de análisis de sentimiento se puede ver en la siguiente imagen

Fase de limpieza

Una vez que tenemos el conjunto de datos lo primero que hacemos es limpiar los textos de las críticas de etiquetas HTML ya que los datos provienen de una página web.

El siguiente paso sería quedarnos con las palabras del texto, es decir eliminar todo lo que no sean letras(números, comillas, comas...). Esto lo hacemos utilizando expresiones regulares mediante la librería re. Antes de realizar este paso hemos considerado diferentes caminos para obtener información del texto que no sean letras. Para ello hemos imprimido bastantes críticas para ver qué cosas podríamos hacer.

- Separar las palabras negadas abreviadas como “don’t” o “aren’t” en las palabras correspondientes “do not” y “are not” porque creemos que las palabras negativas pueden dar información sobre si una crítica

es negativa o positiva (por ejemplo, una crítica con muchas palabras negativas). Para ello se ha utilizado expresiones regulares y se ha notado una pequeña mejora en los resultados de los modelos.

- Se ha pensado en sustituir las exclamaciones "¡!" e interrogantes "¿?" por palabras del tipo "exclamation" o "question" pero tras visualizar algunas críticas se ha observado pocas y estaban indistintamente en criticas positivas y negativas. Aun si, ante la dificultad de ver todas las criticas hemos implementado en Python utilizando expresiones regulares, pero no se han obtenido mejoras en los modelos por lo que se ha desechado esta opción.
- Se ha pensado en obtener información de los números que aparecen en los textos. Un vistazo nos permite observar que hay bastantes números en los textos, números sueltos y años. La primera idea ha sido sustituir estos números por una palabra del tipo "number" pero no se han obtenido mejores resultados en los modelos. Sin embargo, al probar dejando los números tal cual (sin sustituirlos por alguna palabra) se ha observado una pequeña mejora por lo tanto se ha procedido a dejar los números en los textos.
- Otra opción que se ha considerado es la de corregir palabras utilizando una librería para ello, pero no se obtienen mejoras en los resultados en mi opinión debido a que estas palabras se repiten muy poco en el texto. Se descarta esta opción también.
- Se han observado bastantes nombres de personas (de personajes de las películas). Se ha optado por sustituir estos nombres, utilizando un corpus de NLTK para detectarlos, por una palabra del tipo "character" o eliminarlos, pero ninguno de estos procesos ayuda a mejorar los resultados de los modelos.
- Otra opción fue la de detectar insultos en las criticas (las críticas con insultos suelen ser negativas, aunque hay excepciones). Una primera idea fue sustituir estos insultos por una palabra "insult" pero observamos que había varios insultos que eran muy comunes y se perdía información. Así que el siguiente paso fue sustituir solo los insultos menos comunes, pero aun así no se obtenían mejores resultados en los modelos.

Todas estas ideas pueden ser aplicadas en otras tareas de nlp y dependiendo del texto que estemos manejando pueden ser útiles o no.

Stopwords

Una vez considerados estos pasos nos quedamos con las palabras y números de los documentos, los tokenizamos(dividir los documentos en palabras dentro de una lista) usando la función .split() y convertimos todas las palabras del texto en minúsculas.

El siguiente paso que hemos considerado es la eliminación de stopwords. Los stopwords o palabras vacías son palabras sin significado como artículos, pronombres, preposiciones, etc. que son filtradas antes o después del procesamiento de datos en lenguaje natural (texto).

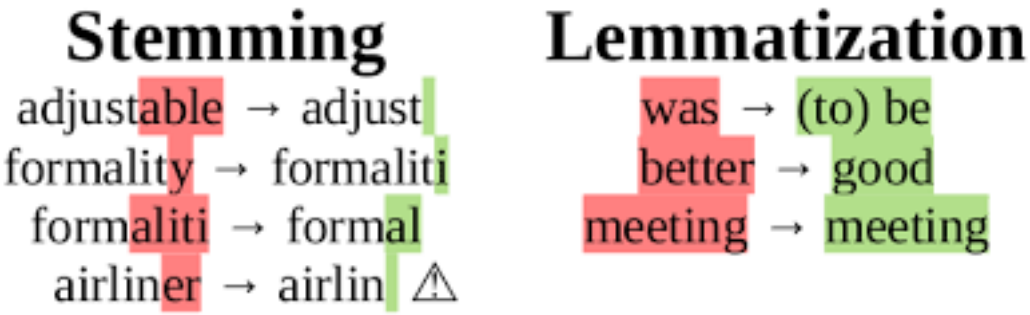
Aquí hay que tener cuidado respecto a la tarea específica de nlp que se quiere resolver. Lo primero que hemos hecho ha sido imprimir la lista de stopwords de NLTK para observar si había alguna palabra que pensáramos que podía aportar información. En efecto se ha observado que hay algunas palabras con negaciones como "arent" o "doesnt". Se ha decidido sacar estas palabras de la lista de stopwords.

Lemmatization y stemming

Stemming es un método para reducir una palabra a su raíz('stem' en inglés). El algoritmo más común es el algoritmo de Porter. Existen más métodos. Nosotros hemos probado con este y otro llamado Lancaster.

El lema de una palabra es la forma que se acepta como representante de todas las formas flexionadas de dicha palabra. Por ejemplo, el lema de las palabras "" y "" es "". La lematización es el proceso lingüístico que consiste en, dada una forma flexionada, obtener el lema correspondiente

En la siguiente imagen se puede ver ejemplos de estos procesos



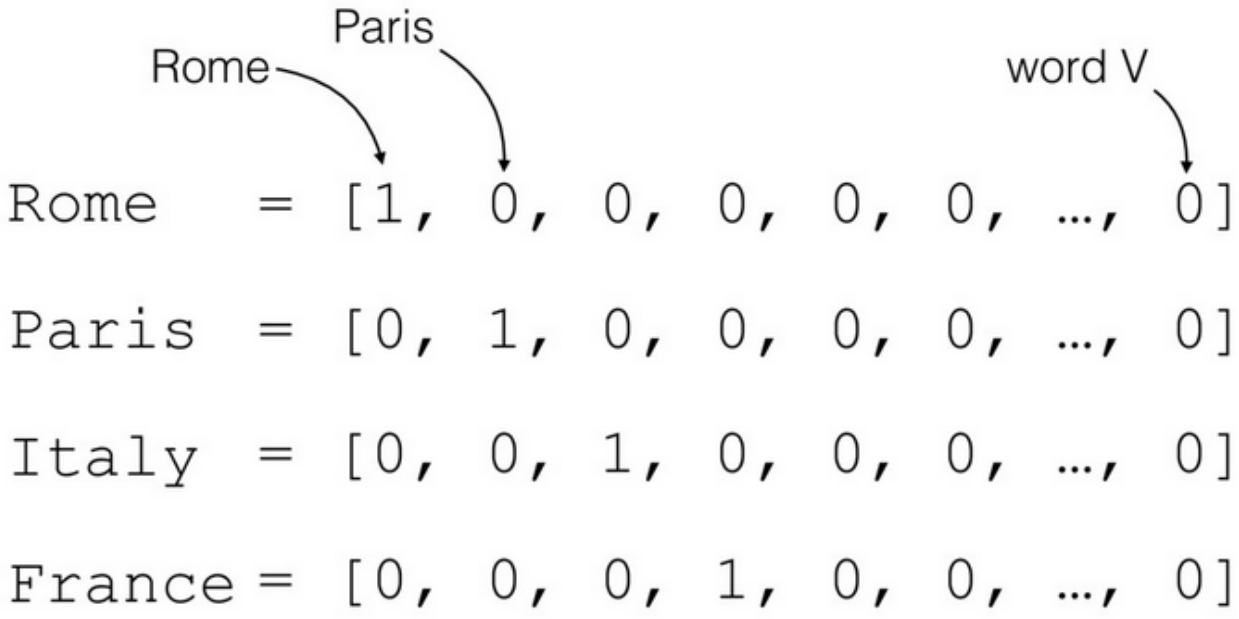
En nuestro caso se han probado ambos procesos obteniendo mejores resultados en los modelos el proceso de lematización.

Fase ve vectorizacion

Queremos alimentar los algoritmos con texto. Este puede ser dividido en frases o palabras pero necesitamos una representación numérica de la palabra. Existen varias maneras para codificar este texto en números.

One hot encoding

Una opción básica es la que se conoce como 'One hot encoding' que es una representación de variables categóricas como vectores binarios. Cada palabra es representada como un vector binario con todos los valores 0 salvo el índice de la palabra en el vector que tendra un 1.



Esta manera de codificar el texto es muy parecida a la codificación en variables dummy que se utiliza en algunos algoritmos de aprendizaje supervisado por ejemplo para convertir las variables categóricas explicativas que hay en un modelo en variable numéricas.

Como vemos en la imagen cada palabra se representa en un espacio multidimensional del tamaño del vocabulario (en el ejemplo v) Tendremos v variables explicativas (cada palabra) de n muestras u observaciones(cada documento o critica)

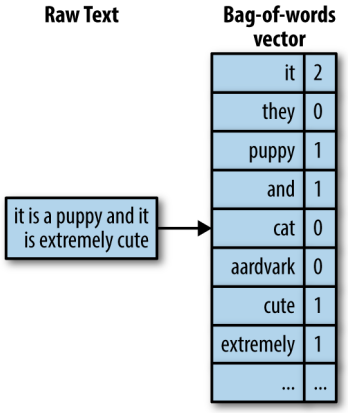
Sin embargo, cuando se quiere vectorizar cada palabra de un corpus de documentos este método no es muy eficiente pues se obtendrá una matriz enorme de datos y el 99% de los elementos de ésta serán ceros. A este fenómeno se le conoce como matriz dispersa ('sparse matrix' en inglés) y puede ser problemático. Además de esto encontramos varias desventajas como pueden ser:

- No se tiene en cuenta el orden en el que aparecen las palabras en el documento.
- No se tiene en cuenta la semántica de las palabras.
- No se toma en cuenta el significado de la palabra, el contexto.
- No se toma información sobre la frecuencia de las palabras.

Una idea que mejora este método es el llamado 'bag of words'. Es básicamente lo mismo que la codificación 'one hot' salvo que en este caso se guarda información sobre la ocurrencia de una palabra en cada documento.

Bag of words

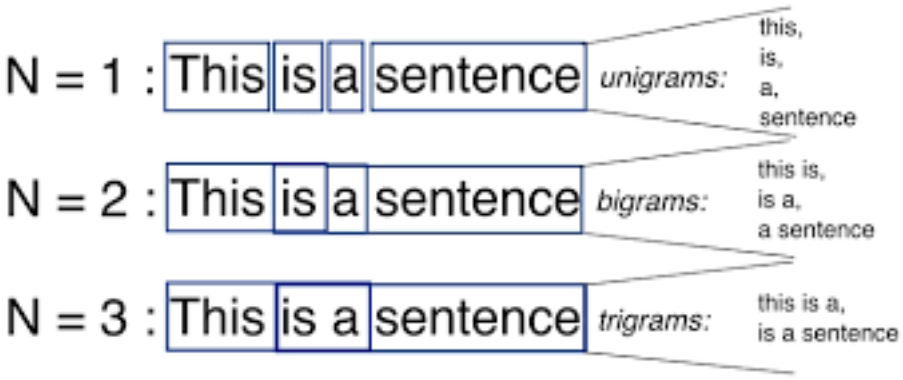
La codificación 'bag of words' es una representación de texto que describe la ocurrencia de las palabras dentro de un documento.



Las diferentes palabras de todos los documentos o críticas forman el vocabulario de tamaño n. Cada documento representa una muestra (fila de la matriz) y cada palabra o token representa cada columna de la matriz.

En este método, al igual que en 'one-hot', cada palabra se representa en un espacio multidimensional del tamaño del vocabulario donde cada palabra representa un vector que apunta en una sola dimensión.

No se toma en cuenta el significado de la palabra, ni el orden en la que aparecen las palabras. Tampoco se tiene en cuenta el contexto, por ejemplo, las frases 'The movie was good, not bad at all' y 'The movie was bad, not good at all' tienen significado opuesto, sin embargo tienen las mismas palabras así que su codificación 'bag of words' será igual. Para solucionar esto es posible utilizar el parámetro n_grams que recoge informacion del contexto.



Bag of words: TF-IDF

Este método se puede ver como una mejora del método bag of words usual

TF-IDF es una medida numérica que expresa la relevancia de una palabra para un documento en una colección. El valor tf-idf aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia de la palabra en la colección de documentos, lo que permite manejar el hecho de que algunas palabras son generalmente más comunes que otras.

Tf-idf es el producto de dos medidas, frecuencia de término y frecuencia inversa de documento. Existen varias maneras de determinar el valor de ambas. En el caso de la frecuencia de término tf(t, d), la opción más sencilla es usar la frecuencia bruta del término t en el documento d, o sea, el número de veces que la palabra t aparece en el documento d

La frecuencia inversa de documento es una medida de si el término es común o no, en la colección de documentos. Se obtiene dividiendo el número total de documentos por el número de documentos que contienen el término, y se toma el logaritmo de ese cociente:

$$\mathrm{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

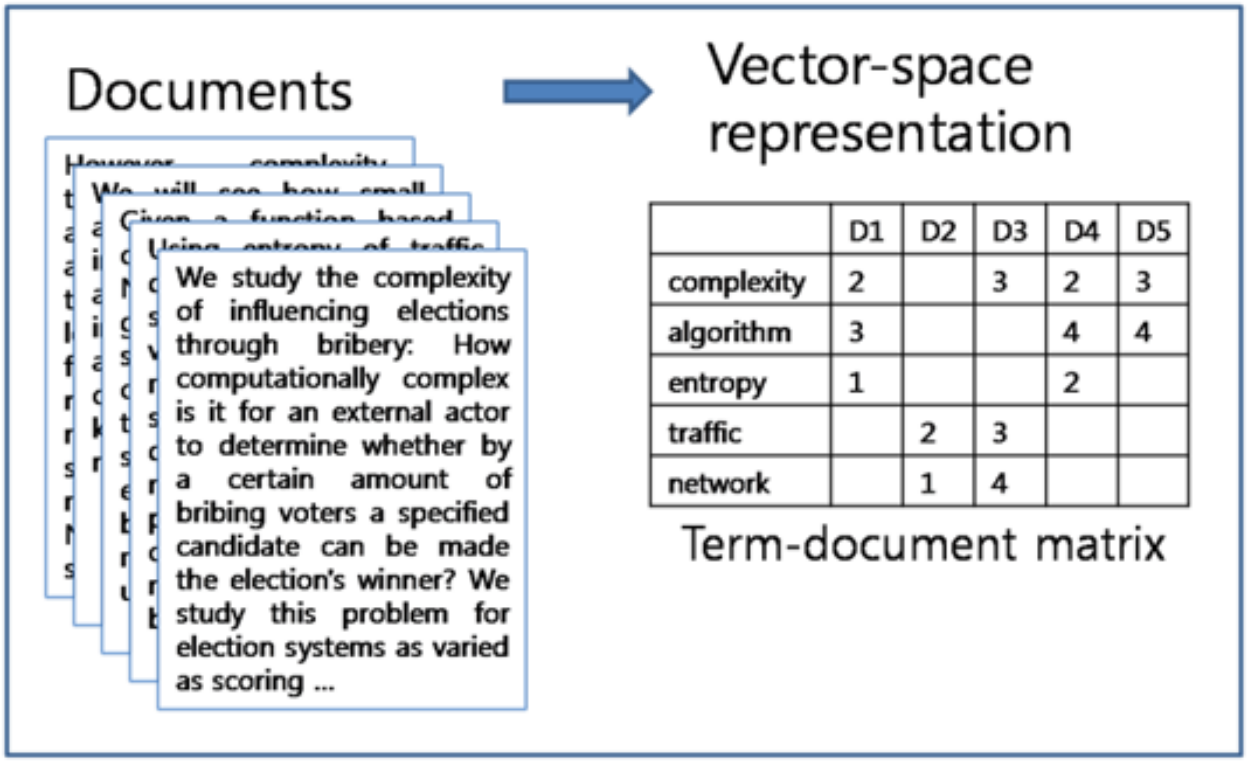
donde

$|D||D|$: cardinalidad de D, o número de documentos en la colección. $|\{d \in D : t \in d\}||\{d \in D : t \in d\}|$: número de documentos donde aparece el término t. Si el término no está en la colección se producirá una división-por-cero. Por lo tanto, es común ajustar esta fórmula a $1 + |\{d \in D : t \in d\}|1 + |\{d \in D : t \in d\}|$.

Luego, tf-idf se calcula como:

$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \times \mathrm{idf}(t, D)$ Un peso alto en tf-idf se alcanza con una elevada frecuencia de término (en el documento dado) y una pequeña frecuencia de ocurrencia del término en la colección completa de documentos. Como el cociente dentro de la función logaritmo del idf es siempre mayor o igual que 1, el valor del idf (y del tf-idf) es mayor o igual que 0. Cuando un término aparece en muchos documentos, el cociente dentro del logaritmo se acerca a 1, ofreciendo un valor de idf y de tf-idf cercano a 0.

En una matriz documento-término las filas representan los documentos y las columnas representan los términos o palabras mientras que la matriz término-documento es la traspuesta.



	D1	D2	D3	D4	D5
complexity	2		3	2	3
algorithm	3			4	4
entropy	1			2	
traffic		2	3		
network		1	4		

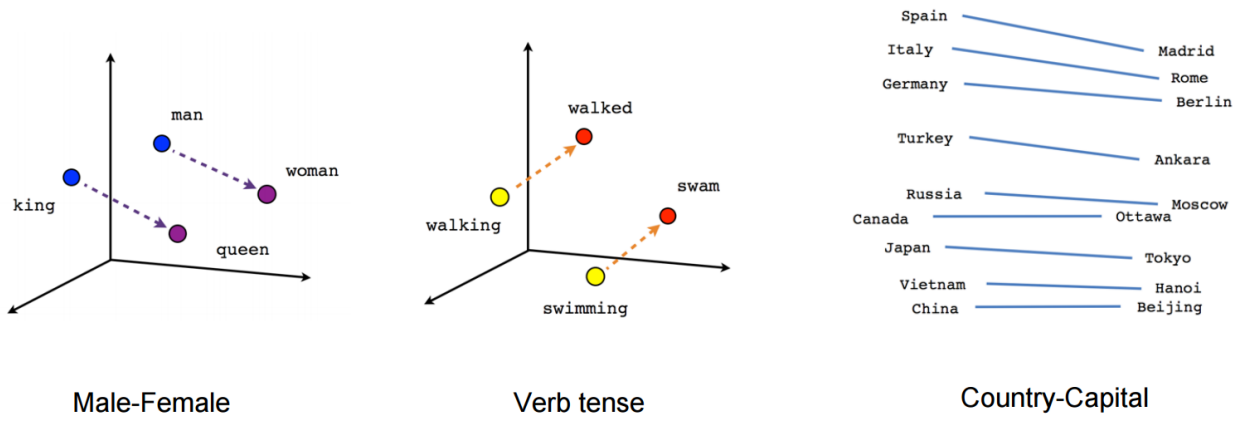
Esta idea tiene sus ventajas respecto al metodo anterior y se han visto resultados muy buenos en una gran variadad de tareas. Sin embargo para tareas de nlp mas complejas como traduccion de texto o generacion de texto sigue sin tener en cuenta aspectos importantes mencionados anteriorment para la codificacion "bag of words" usual.

Word2vec o Word embbeding

Hasta ahora en los tipos de codificación que hemos visto cada palabra se representa en un espacio multidimensional del tamaño del vocabulario y representa un vector que apunta en una sola dimensión. Es decir que, si tenemos tres palabras "dog", "cat" and "game" la distancia entre los vectores de estas palabras es la misma pero conceptualmente podemos entender que las palabras "dog" y "cat" tienen mayor relación que la que pueden tener "dog" y "game" o "cat" y "game". La idea es usar la dirección de los vectores para codificar cierta relación que existe entre estas palabras

Las palabras embebidas ("word embbedings" en inglés) son vectores densos de valores coma flotante que contienen información de la semántica de las palabras. Nosotros no tenemos que hacer esta codificación a mano, sino que sería aprendida por un algoritmo de aprendizaje no supervisado que se encarga de convertir las palabras en vectores y que el vector codifique la relación existente entre esas palabras.

Podemos ver un ejemplo de alguna de estas relaciones.



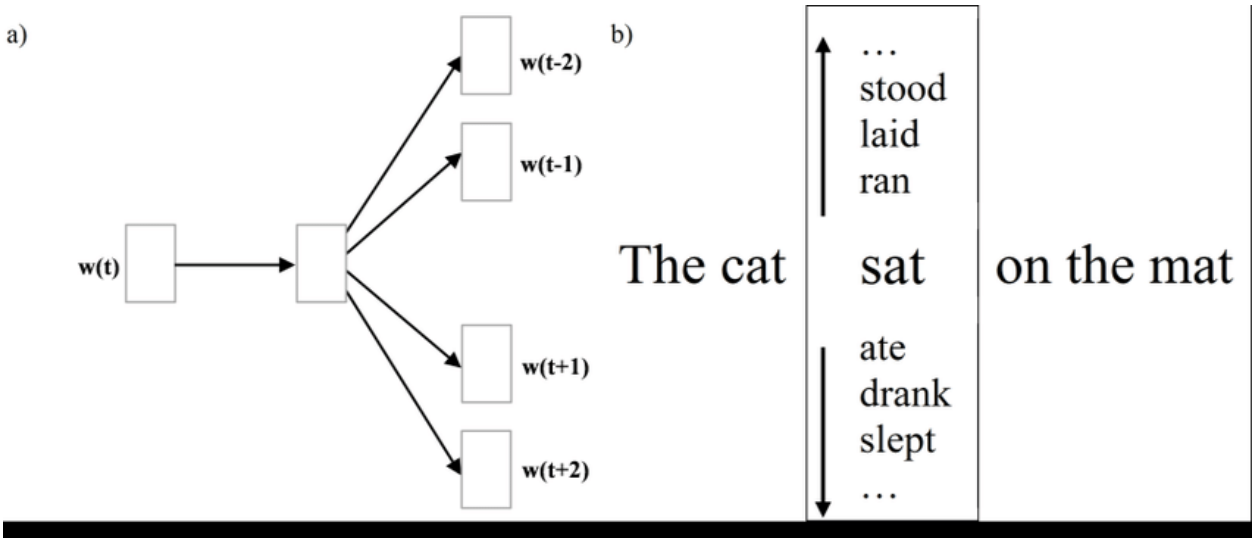
Male-Female Verb tense Country-Capital

Básicamente, queremos usar las palabras vecinas que representan una palabra objetivo utilizando una red neuronal cuya capa oculta codifica la representación de la palabra.

Existen dos tipos de algoritmos:

- CBOW (Continuous Bag of words): la idea es predecir la probabilidad de una palabra dado cierto contexto.
- Skip-Gram model: la idea es predecir el contexto dada una palabra (el input es la palabra objetivo, mientras que los outputs son las palabras vecinas de la palabra objetivo).

En el modelo Skip-Gram los datos de entrada y salida son de la misma dimensión y codificados "one-hot". La contiene una capa oculta cuta dimensión es igual al tamaño del embedding. A la salida de la capa oculta se le aplica una función de activación a cada elemento del vector salida que describe como de probable una palabra especifica aparecerá en el contexto. En el siguiente grafico se visualiza la estructura de la red.



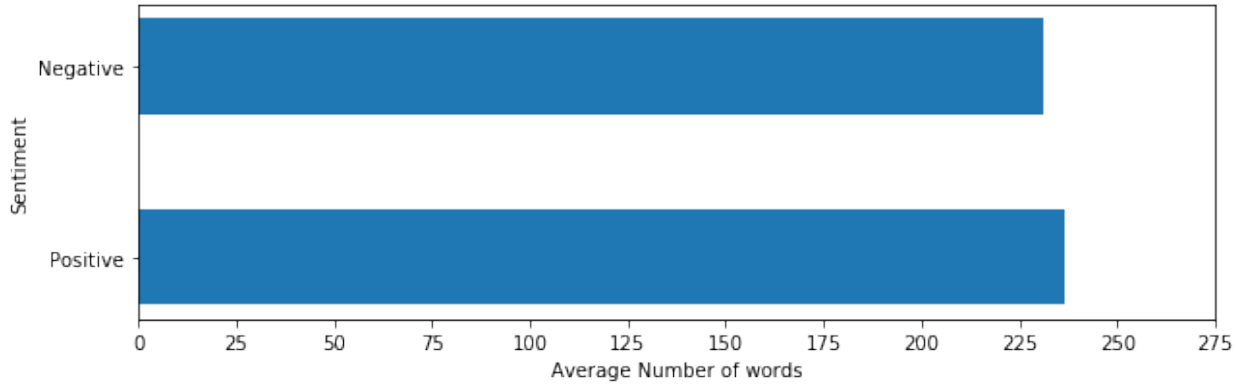
Aunque esta es una codificación muy potente que se viene utilizando mucho durante los últimos años también tiene limitaciones: por ejemplo, no es capaz de captar la relación de las palabras dentro de una frase.

Se quiere un modelo que sea capaz de entender la semántica de las palabras y además como se relacionan entre ellas dentro de un texto. Esto se le conoce como un modelo del lenguaje cuya tarea es predecir cual es la siguiente palabra en función de las palabras anteriores (semántica y estructura del lenguaje). El uso de redes neuronales recurrentes ha tenido un gran avance en este sentido durante los últimos tiemposcon algoritmos capaces de comprender la estructura completa del lenguaje.

Una observación muy interesante acerca de los modelos del lenguaje o los modelos word2vec es que este tipo de conocimiento puede ser generalizable y podría usarse para resolver otro tipo de tareas. Este proceso de entrenar un modelo ya pre-entrenado para resolver una tarea específica se conoce como "fine-tunning" y puede dar resultados muy buenos con un entrenamiento menor y un coste más bajo. De hecho, esta idea ha posibilitado grandes avances en el campo del nlp en los últimos dos años.

Fase de entrenamiento del modelo

Antes de entrenar el modelo hemos comprobado que el dataset de entrenamiento esta balanceado, es decir, que el numero de críticas positivas y negativas es aproximadamente el mismo.



Regresión logística

Introducción

Los modelos de regresión logística son una herramienta que permite explicar el comportamiento de una variable respuesta discreta (binaria o con más de dos categorías) a través de una o más variables independientes explicativas de naturaleza cuantitativa y/o cualitativa. Según el tipo de variable respuesta estaremos hablando de regresión logística binaria (variable dependiente con 2 categorías), o de regresión logística multinomial (variable dependiente con más de 2 categorías). En nuestro caso la variable respuesta es binaria por lo que aplicaremos la regresión logística binaria.

Los modelos de respuesta discreta son un caso particular de los modelos lineales generalizados al igual que los modelos de regresión lineal o el análisis de la varianza

La regresión logística es uno de los algoritmos más utilizados en Machine Learning.

Tenemos una variable respuesta o dependiente Y que toma dos valores, Y = 1 (indica si una crítica es positiva) e Y = 0 (indica si la crítica es negativa). Denotemos por n el numero

de variables independientes del modelo representadas por $X = (X_1, X_2, ..., X_n)$

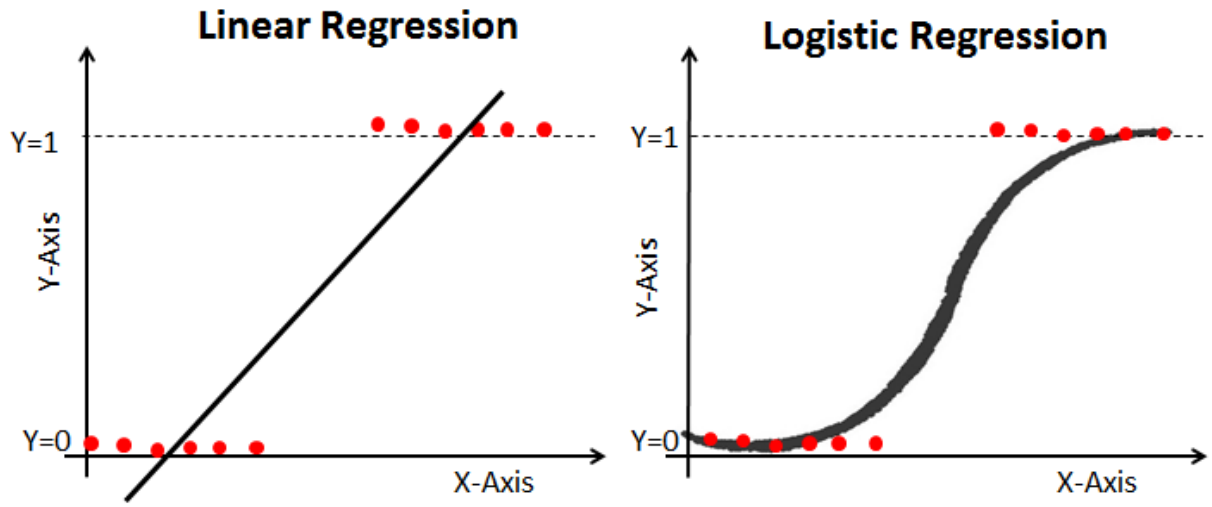
La formulación genérica del modelo de regresión logística para modelar la probabilidad de ocurrencia de un suceso seria $Y = p_x + \epsilon$ donde ϵ es el termino de error, p_x es la probabilidad de que la respuesta Y tome el valor 1 para el valor observado x y se modeliza como:

$$P(Y = 1|X = x) = p_x = \frac{exp(\beta_0 + \sum \beta_i x_i)}{1 + exp(\beta_0 + \sum \beta_i x_i)}$$

siendo $x = (x_1, x_2, ..., x_n)$ un valor observado de las variables explicativas.

Si aplicamos una transformación logit a la ecuación anterior, obtenemos un modelo de regresión lineal que facilitara la posterior interpretación del modelo:

$$logit(p_x) = log(\frac{p_x}{1 - p_x}) = \beta_0 + \sum \beta_i x_i$$



Estimación

Para una regresión logística binaria los datos se presentan no agrupados $(x_1, y_1), ..., (x_n, y_n)$. La variable respuesta se considera agrupada suponiendo que sigue una distribución Bernoulli $y_j - B(1, p_j)$.

Los parámetros desconocidos β son estimados usando el método de máxima verosimilitud, que consiste en proporcionar la estimación que otorgue máxima probabilidad o verosimilitud a los datos observados. Se toma la función de verosimilitud y se maximiza (su logaritmo) derivando respecto a cada uno de los parámetros e igualando a cero se obtienen las ecuaciones de verosimilitud.

Observación: los estimadores MV de un modelo siempre existen y son únicos (salvo en ciertos casos de separación completa) por lo que será necesario que exista cierto solapamiento en los datos para la existencia de estos parámetros.

Las ecuaciones obtenidas no son lineales en los parámetros por lo que se requieren métodos iterativos como el método de Newton-Raphson para su resolución.

Finalmente, la estimación máximo verosímil de p_j viene dada por:

$$\hat{p}_j = \frac{exp(\hat{\beta}_0 + \sum \hat{\beta}_i x_i)}{1 + exp(\hat{\beta}_0 + \sum \hat{\beta}_i x_i)}, j = 1, ..., n$$

siendo $\hat{\beta}$ los estimadores MV de los parámetros β. Esta estimacion p_j se corresponde con la estimación de la respuesta $\hat{y}_j$

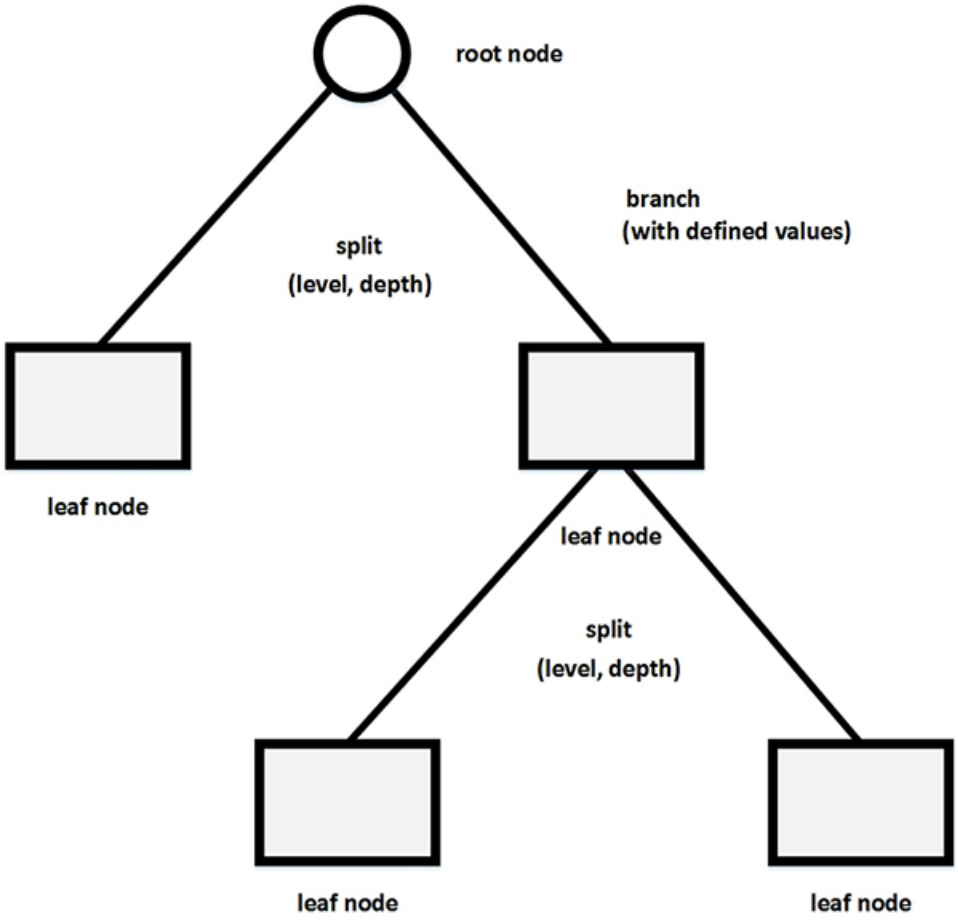
Bondad de ajuste

Una vez construido el modelo de regresión logóstica simple, tiene sentido comprobar como de bueno es el ajuste de los valores predichos por el modelo a los valores observados. Existen diversas formas de medir la bondad de ajuste de un modelo de regresiòn logística. De forma global, puede ser evaluada a traves de medidas tipo tasa de clasificaciones correctas o a traves de una serie de test estadísticos.

Arboles de decision

Para entender este modelos es necesario entender que es una arbol de decisión. Un árbol de decisión es un modelo de predicción utilizado en diversos ámbitos que van desde la inteligencia artificial hasta la Economía. Dado un conjunto de datos se fabrican diagramas de construcciones lógicas, muy similares a los sistemas de predicción basados en reglas, que sirven para representar y categorizar una serie de condiciones que ocurren

de forma sucesiva, para la resolución de un problema. Estos arboles se pueden utilizar tanto para tareas de regresion como para tareas de clasificacion, y también pueden contener predictores tanto cuantitativos como cualitativos. Una de las ventajas de los árboles de decisión en comparación con otros métodos es su fácil interpretación y útil representación gráfica. Estos arboles estan formados por nodos, ramas, hojas...



Basic Parts of a Decision Tree

A la hora de construir un arbol de regresión se sigue los sigientes pasos: - División del espacio del predictor en regiones rectangulares de múltiples dimensiones. El objetivo es encontrar las regiones $R_1, ..., R_j$ que minimicen la suma de los cuadrados de los residuos("Residual Sum of Squares o RSS" en ingles) - Para cada una de las observaciones pertenecientes a la región R_j se asigna la misma predicción, que es simplemente la media de la variable respuesta de las observaciones de entrenamiento en R_j.

Resulta computacionalmente inviable considerar todas las posibles particiones del predictor, por lo que se opta por una división binaria recursiva ("recursive binary splitting""), similar al concepto de stepwise selection, consiguiendo también obtener buenos resultados. El proceso comienza en el punto más alto del árbol (donde todas las observaciones perteneces a una misma región), y de manera sucesiva se divide el espacio del predictor, donde cada división genera dos nuevas ramas. Se optará además por la mejor división en cada paso, sin tener en cuenta si dicha división mejorará el árbol en futuros pasos o divisiones.

El proceso continúa, pero esta vez en lugar de dividir el espacio completo del predictor, se divide una de las dos regiones creadas en el paso anterior. El proceso continúa hasta que se alcance un criterio de parada, por ejemplo, hasta que ninguna región contenga más de n observaciones, hasta alcanzar un máximo de nodos terminales, etc. Una vez las regiones R1,...,Rj han sido creadas, predecimos la respuesta para una determinada observación de test como la media de las observaciones de entrenamiento en la región en la cual

pertenece dicha observación.

El proceso de división binaria recursiva puede conseguir buenas predicciones con los datos de entrenamiento, ya que reduce el training RSS, lo que implica un sobreajuste a los datos (derivado de la facilidad de ramificación y posible complejidad del árbol resultante), reduciendo la capacidad predictiva para nuevos datos.

Una posible alternativa supone construir un árbol hasta el punto en el que la reducción del RSS debido a cada división supere un determinado límite (alto). Con esta estrategia obtendremos árboles más pequeños, con menos ramificaciones, con lo que conseguimos reducir la varianza y mejorar la interpretabilidad a expensas de una pequeña reducción del bias.

Concretamente, la estrategia consistirá en construir un árbol grande T_0 y luego podarlo (pruning) para obtener un sub-árbol que consiga el menor test error, obtenido mediante validación cruzada o validación simple. Sin embargo, suele resultar muy costoso obtener el error de validación para cada posible sub-árbol debido al gran número de posibles sub-árboles, por lo que se opta por seleccionar un grupo pequeño de sub-árboles a considerar. Una manera de conseguir esto es mediante el cost complexity pruning

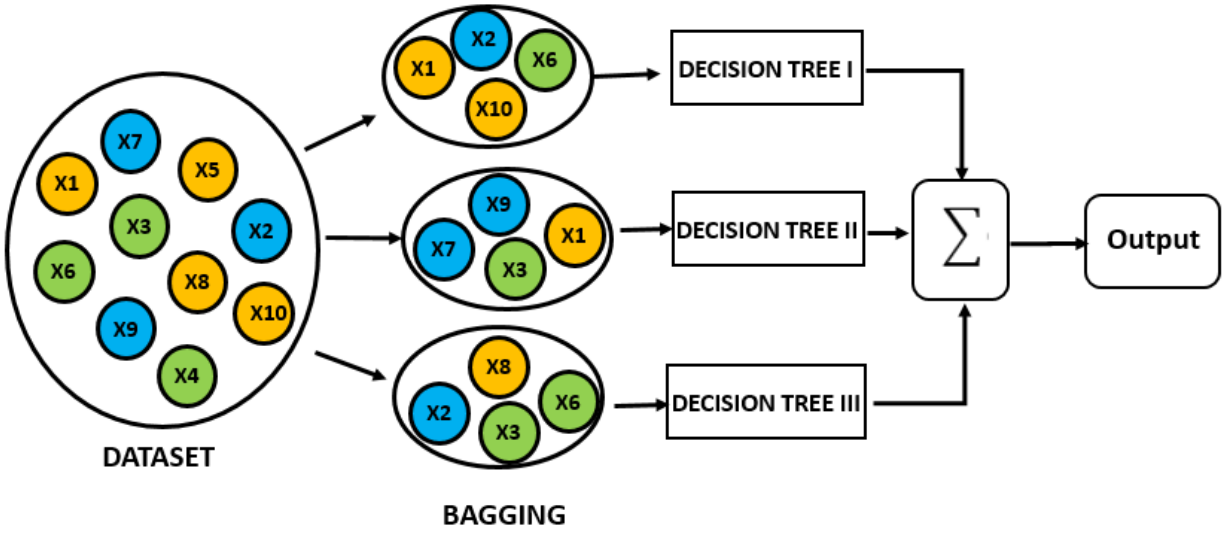
Los árboles de clasificación son muy similares a los de regresión, con la diferencia de que se usan para predecir una variable respuesta cualitativa, asignando la predicción para cada observación como la clase más común (moda) de observaciones de entrenamiento en la región o nodo terminal al que pertenece dicha observación de test. Al igual que con los árboles de regresión, se aplica la división binaria recursiva para generar el árbol, pero el RSS no puede usarse como criterio para estas divisiones, sino otras alternativas como: error de clasificacion, indice de Gini o el indice cross-entropy. Una desventaja de los arboles de decision es que su capacidad de prediccion es superable por otros metodos.Sin embargo, mediante la agregación de varios árboles de decisión, métodos como bagging, random forests y boosting, la capacidad predictiva de los árboles puede mejorarse sustancialmente.

Los métodos de bagging, random forests y boosting nos permiten mejorar sustancialmente el rendimiento predictivo de modelos basados en árboles, aunque con el inconveniente de una considerable reducción en la facilidad de interpretación del modelo final. Estos métodos también se conocen como métodos de ensemble o métodos combinados, que son los que utilizan múltiples algoritmos de aprendizaje para obtener predicciones que mejoren las que se podrían obtener por medio de cualquiera de los algoritmos individuales. Son aplicables a muchos métodos de aprendizaje estadísticos (no solo árboles de decisión) para regresión o clasificación.

Bagging

Los árboles de decisión sufren de alta varianza, lo que significa que, si dividiéramos al azar los datos de entrenamiento en dos grupos y ajustáramos un árbol de decisión a cada mitad, los resultados que obtendríamos podrían ser bastante diferentes. Por el contrario, un procedimiento o método con baja varianza dará resultados parecidos aun aplicándose sobre sets de datos distintos. El método de bagging o bootstrap aggregation es un procedimiento utilizado para reducir la varianza de un método de aprendizaje estadístico.

En un problema de clasificación, existen varias posibilidades para aplicar bagging, pero la más simple es la siguiente: dada una observación de test, podemos obtener la clase predicha por cada uno de los B árboles, y escoger como predicción final para dicha observación la clase más común de entre las B predicciones (predicción de cada árbol).



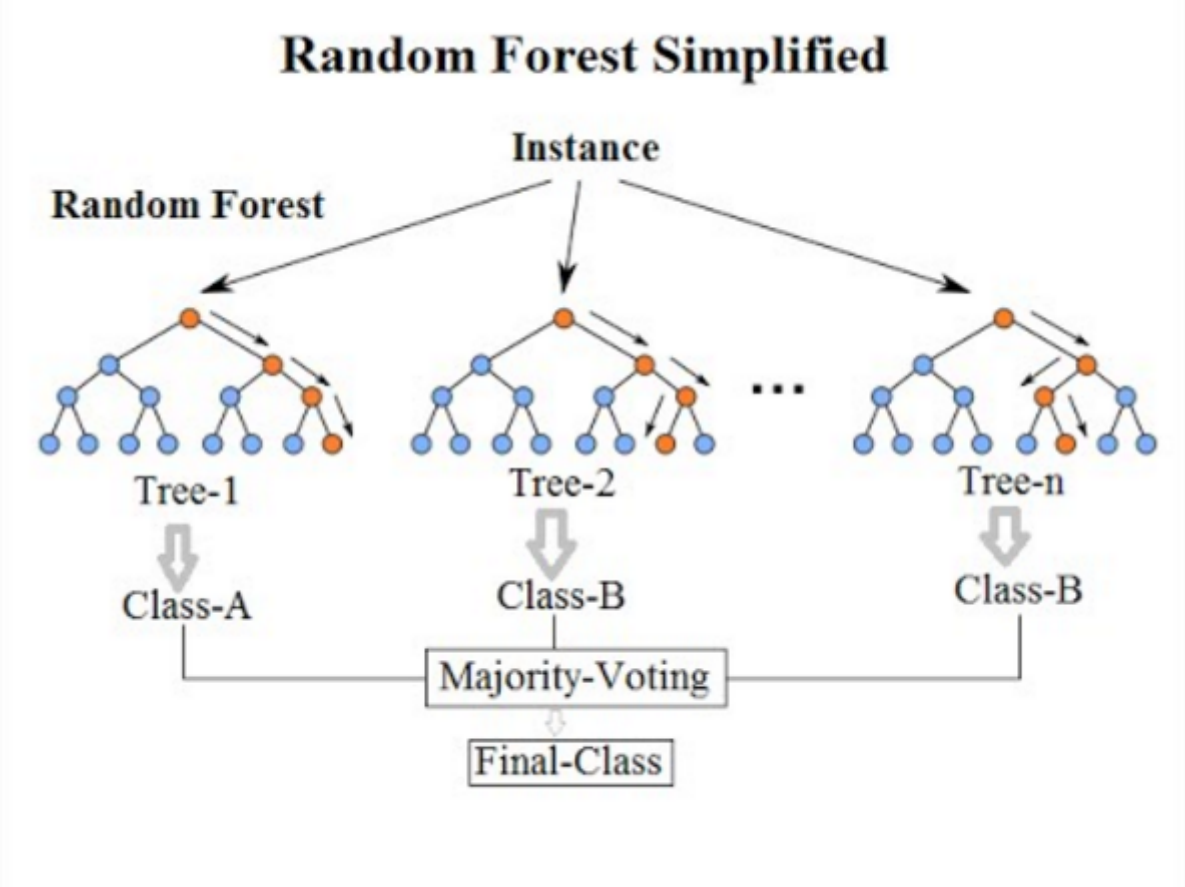
El número de árboles (B) a crear no es un parámetro crítico a la hora de aplicar bagging. Ajustar un gran número de árboles no aumentará el riesgo de overfitting, por lo que usaremos un número lo suficientemente alto como para alcanzar la estabilización en la reducción del test error.

Una manera directa de estimar el test error de un modelo al que se aplica bagging sin necesidad de aplicar la validación cruzada o simple: en promedio, cada árbol generado por bootstrapping usa en torno a 2/3 de las observaciones. El 1/3 restante de observaciones no usadas para ajustar cada árbol se conocen como out-of-bag (OOB). Por lo tanto, podemos obtener la predicción para la i-ésima observación de test usando cada uno de los árboles en los cuales dicha observación sea OOB.

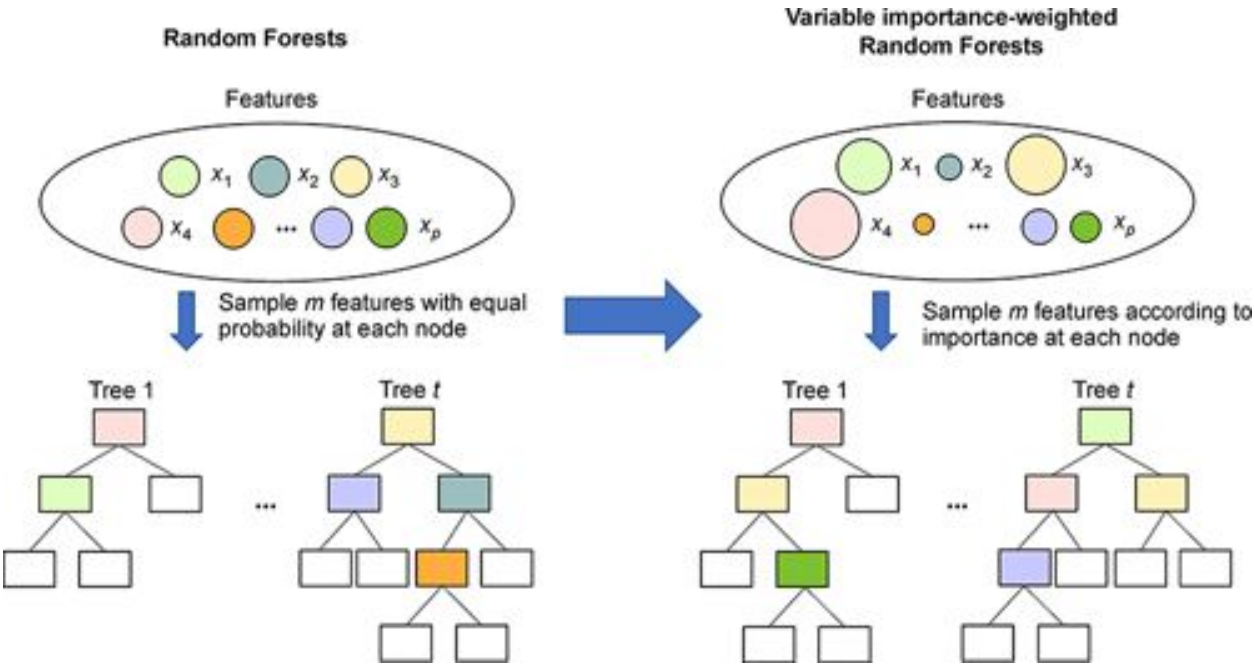
Random Forest

Supóngase que se cuenta con un set de datos cuenta con un predictor muy importante o influyente que destaca sobre el resto. En este caso, todos o casi todos los árboles generados por bagging usarán este predictor en la primera ramificación, por lo que acabarán siendo similares unos a otros, y las predicciones entre ellos estarán altamente correlacionadas. En este escenario, la aplicación de bagging promediando valores correlacionados no consigue una reducción sustancial de la varianza con respecto a un solo árbol.

El método de random forests proporciona una mejora a los árboles combinados por bagging en cuanto a que los decorrelaciona, teniendo en cuenta solo un subgrupo de predictores en cada división. Al igual que en el bagging, se construyen un número de árboles de decisión a partir de pseudo-muestras generadas por bootstrapping. Esta vez, se escogen de entre todos los p predictores una muestra aleatoria de m predictores como candidatos antes de cada división, generalmente $m???\sqrt{p}$ (si m = p, bagging y random forests darían resultados equivalentes)



¿Que variables influyen mas en la variable respuesta? Mientas que en el modelo de regresion logistica podemos observar la significatividad de las variables regresoras, los arboles de decision no se basan en contrastes de hipotesis pero en el modelo random forest existe otro camino para saber que variables estan mas relacionadas con la respuesta. La siguiente imagen explica esta idea.



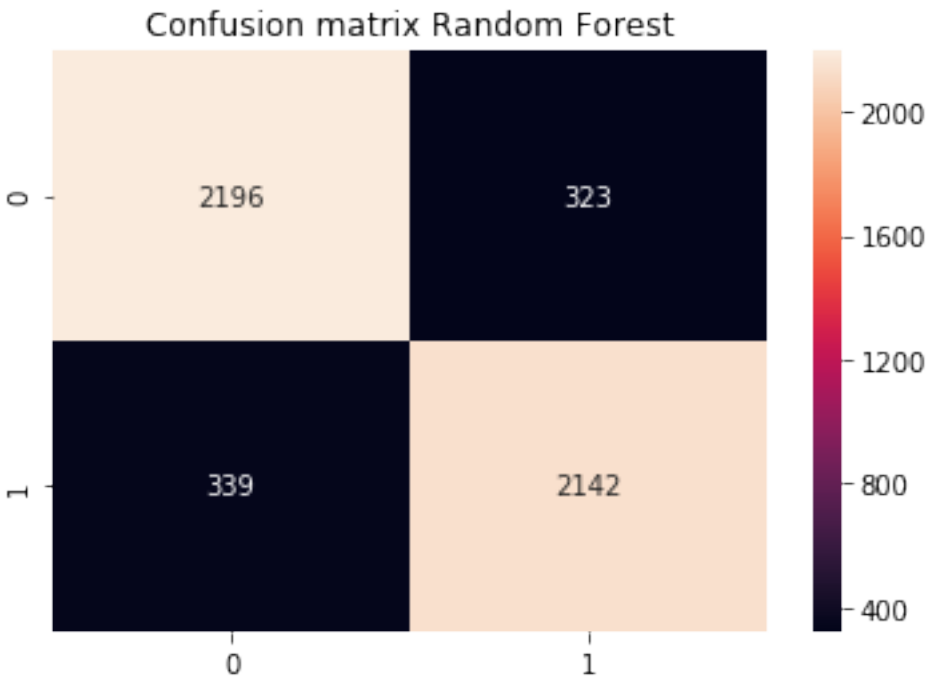
Comparativa de los modelos

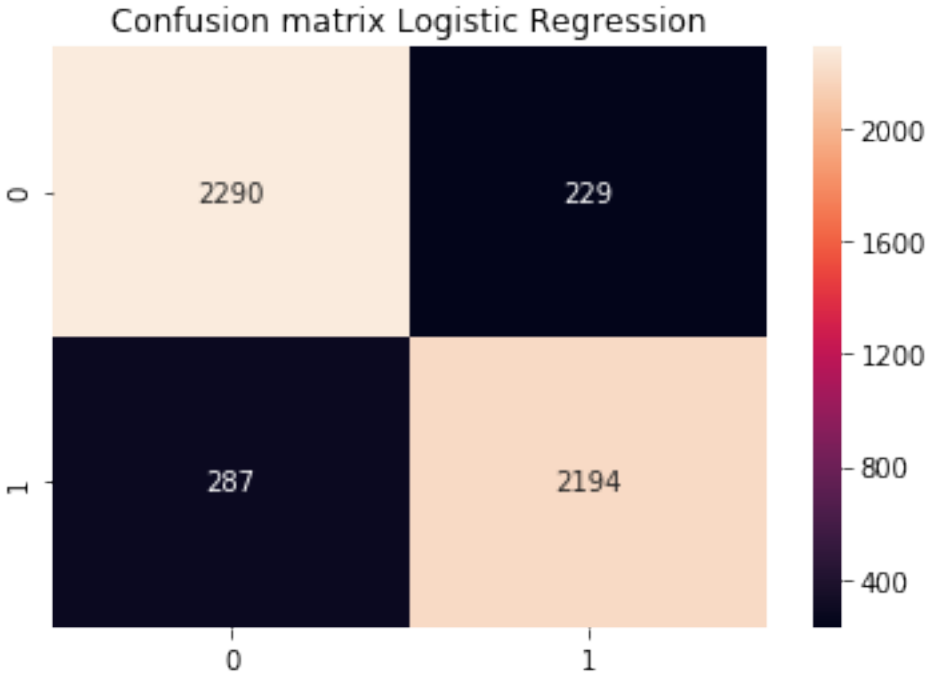
Hemos aplicado los algoritmos de regresión logística y random forest. Para evaluar y comparar los modelos hemos utilizado la matriz de confusion. Esta matriz tiene la siguiente estructura:

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

- VP es la cantidad de positivos que fueron clasificados correctamente como positivos por el modelo.
- VN es la cantidad de negativos que fueron clasificados correctamente como negativos por el modelo.
- FN es la cantidad de positivos que fueron clasificados incorrectamente como negativos.
- FP es la cantidad de negativos que fueron clasificados incorrectamente como positivos.

Como el conjunto de test no viene etiquetado hemos dividido el conjunto de entrenamiento en dos partes, un dataset de entrenamiento y otro de test para poder realizar la evaluacion. Logicamente para los resultados finales hemos utilizado todo el conjunto de entrenamiento para alimentar los modelos.


Confusion matrix Random Forest
2196
323
339
2142
0
1
0
1
2000
1600
1200
800
400


Confusion matrix Logistic Regression
2290
229
287
2194
0
1
0
1
2000
1600
1200
800
400