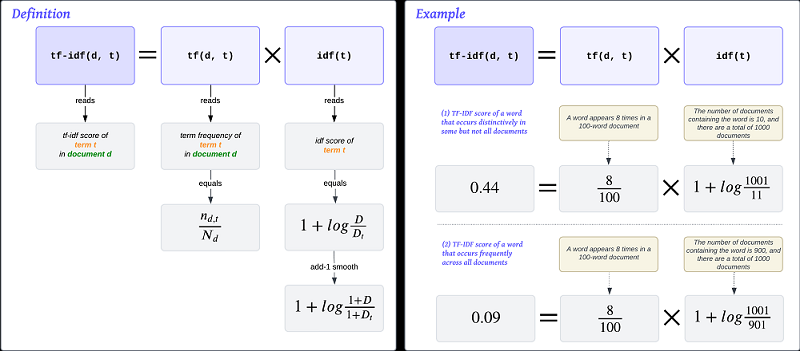
**Python Text Analysis: Bag of Words**

**Term Frequency-Inverse Document Frequency**

So far, we're relying on word frequency to give us information about a document. This assumes if a word appears more often in a document, it's more informative. However, this may not always be the case. For example, we've already removed stop words because they are not informative, despite the fact that they appear many times in a document. We also know the word "flight" is among the most frequent words, but it is not that informative, because it appears in many documents. Since we're looking at airline tweets, we shouldn't be surprised to see the word "flight"!

To remedy this, we use a weighting scheme called **tf-idf (term frequency-inverse document frequency)**. The big idea behind tf-idf is to weight a word not just by its frequency within a document, but also by its frequency in one document relative to the remaining documents. So, when we construct the DTM, we will be assigning each term a **tf-idf score**. Specifically, term $t$ in document $d$ is assigned a tf-idf score as follows:

****

In essence, the tf-idf score of a word in a document is the product of two components: **term frequency (tf)** and **inverse document frequency (idf)**. The idf acts as a scaling factor. If a word occurs in all documents, then idf equals 1. No scaling will happen. But idf is typically greater than 1, which is the weight we assign to the word to make the tf-idf score higher, so as to highlight that the word is informative. In practice, we add 1 to both the denominator and numerator ("add-1 smooth") to prevent any issues with zero occurrences.

We can also create a tf-idf DTM using `sklearn`. We'll use a `TfidfVectorizer` this time:

*from sklearn.feature\_extraction.text import TfidfVectorizer*

*# Create a tfidf vectorizer*

*vectorizer = TfidfVectorizer(lowercase=True,*

*…………………………………..stop\_words='english'*

*…………………………………..min\_df=2,*

*…………………………………..max\_df=0.95,*

*…………………………………..max\_features=None)*

*# Fit and transform*

*tf\_dtm = vectorizer.fit\_transform(tweets['text\_lemmatized'])*

*tf\_dtm*

*<Compressed Sparse Row sparse matrix of dtype 'float64'*

*with 87904 stored elements and shape (11541, 3571)>*

*# Create a tf-idf dataframe*

*tfidf = pd.DataFrame(tf\_dtm.todense(),*

*……………………………columns=vectorizer.get\_feature\_names\_out(),*

*……………………………index=tweets.index)*

*tfidf.head()*

| ***aa*** | ***aadv*** | ***aadvantage*** | ***aal*** | ***abandon*** | ***abc*** | ***ability*** | ***able*** | ***aboard*** | ***abq*** | ***...*** | ***yummy*** | ***yup*** | ***yvonne*** | ***yvr*** | ***yyj*** | ***yyz*** | ***zero*** | ***zone*** | ***zoom*** | ***zurich*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *...* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* |
| *1* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *...* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* |
| *2* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *...* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* |
| *3* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *...* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* |
| *4* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *...* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* |

You may have noticed that the vocabulary size is the same as we saw in Challenge 2. This is because we used the same parameter setting when creating the vectorizer. But the values in the matrix are different—they are tf-idf scores instead of raw counts.

**Interpret TF-IDF Values**

Let's take a look the document where a term has the highest tf-idf values. We'll use the `.idxmax()` method to find the index.

*# Retrieve the index of the document*

*tfidf.idxmax()*

*aa 10077*

*aadv 9285*

*aadvantage 9974*

*aal 10630*

*abandon 7859*

*...*

*yyz 1350*

*zero 2705*

*zone 3177*

*zoom 3920*

*zurich 10622*

*Length: 3571, dtype: int64*

For example, the term "worst" occurs most distinctively in the 918th tweet.

*tfidf.idxmax()['worst']*

*np.int64(918)*

Recall that this is the tweet where the word "worst" appears six times!

*tweets['text\_processed'].iloc[918]*

*"USER is the worst. worst reservation policies. worst costumer service. worst worst worst. congrats, USER you're not that bad!"*

How about "cancel"? Let's take a look at another example.

*tfidf.idxmax()['cancel']*

*np.int64(5945)*

*tweets['text\_processed'].iloc[5945]*

*'USER cancelled flighted 😢'*

**🥊 Challenge 3: Words with Highest Mean TF-IDF scores**

We have obtained tf-idf values for each term in each document. But what do these values tell us about the sentiments of tweets? Are there any words that are  particularly informative for positive/negative tweets?

To explore this, let's gather the indices of all positive/negative tweets and calculate the mean tf-idf scores of words appear in each category.

We've provided the following starter code to guide you:

* Subset the tweets dataframe according to the airline\_sentiment label and retrieve the index of each subset (.index). Assign the index to positive\_index or negative\_index.
* For each subset:
* Retrieve the td-idf representation
* Take the mean tf-idf values across the subset using .mean()
* Sort the mean values in the descending order using .sort\_values()
* Get the top 10 terms using .head()

Next, run pos.plot and neg.plot to plot the words with the highest mean tf-idf scores for each subset.

*# Complete the boolean masks*

*positive\_index = tweets[tweets['airline\_sentiment'] == 'positive'].index*

*negative\_index = tweets[tweets['airline\_sentiment'] == 'negative'].index*

*# Complete the following two lines*

*pos = tfidf.loc[positive\_index].mean().sort\_values(ascending=False).head(10)*

*neg = tfidf.loc[negative\_index].mean().sort\_values(ascending=False).head(10)*

*pos.plot(kind='barh',*

*………….xlim=(0, 0.18),*

*………….color='cornflowerblue',*

*………….title='Top 10 terms with the highest mean tf-idf values for positive tweets');*

*Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.*

*neg.plot(kind='barh',*

*.………….xlim=(0, 0.18),*

*.………….color='darksalmon',*

*.………….title='Top 10 terms with the highest mean tf-idf values for negative tweets');*

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

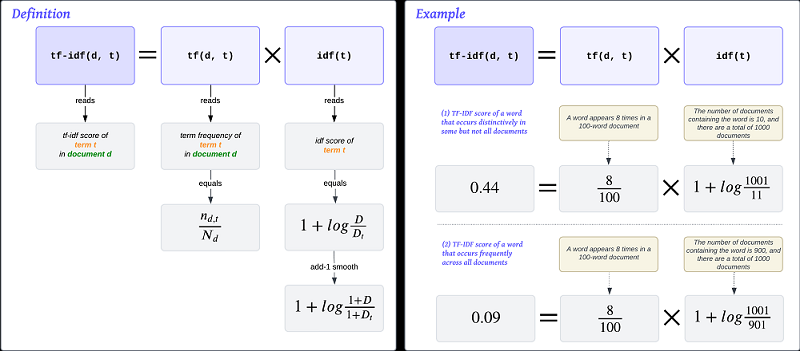
🔔 **Question**: How would you interpret these results? Share your thoughts in the chat!

**Análisis de texto en Python: Bolsa de palabras**

**Frecuencia de términos - Frecuencia inversa de documentos**

Hasta ahora, nos basamos en la frecuencia de palabras para obtener información sobre un documento. Esto supone que si una palabra aparece con más frecuencia en un documento, es más informativa. Sin embargo, esto no siempre es así. Por ejemplo, ya hemos eliminado las palabras vacías porque no son informativas, a pesar de que aparecen muchas veces en un documento. También sabemos que la palabra "vuelo" es una de las más frecuentes, pero no es tan informativa, ya que aparece en muchos documentos. Dado que estamos analizando tweets de aerolíneas, no debería sorprendernos ver la palabra "vuelo".

Para solucionar esto, utilizamos un esquema de ponderación llamado **tf-idf** (término frecuencia-inversa frecuencia de documento). La idea principal de **tf-idf** es ponderar una palabra no solo por su frecuencia dentro de un documento, sino también por su frecuencia en un documento en relación con los demás. Por lo tanto, al construir el **DTM**, asignaremos a cada término un **tf-idf score**. Específicamente, al término **$t$** del documento **$d$** se le asigna un **tf-idf score** de la siguiente manera:

****

En esencia, la puntuación **tf-idf** de una palabra en un documento es el producto de dos componentes: la frecuencia de término **(tf)** y la frecuencia inversa de documento (**idf**). La **idf** actúa como un factor de escala. Si una palabra aparece en todos los documentos, la **idf** es igual a 1. No se produce escala. Sin embargo, la **idf** suele ser mayor que 1, que es el peso que asignamos a la palabra para aumentar la puntuación **tf**-**idf** y destacar su carácter informativo. En la práctica, sumamos 1 tanto al denominador como al numerador (**add**-**1 smooth**) para evitar problemas con cero ocurrencias.

También podemos crear un **DTM tf-idf** usando **sklearn**. En esta ocasión, usaremos **TfidfVectorizer**:

*from sklearn.feature\_extraction.text import TfidfVectorizer*

*# Create a tfidf vectorizer*

*vectorizer = TfidfVectorizer(lowercase=True,*

*…………………………………..stop\_words='english'*

*…………………………………..min\_df=2,*

*…………………………………..max\_df=0.95,*

*…………………………………..max\_features=None)*

*# Fit and transform*

*tf\_dtm = vectorizer.fit\_transform(tweets['text\_lemmatized'])*

*tf\_dtm*

*<Compressed Sparse Row sparse matrix of dtype 'float64'*

*with 87904 stored elements and shape (11541, 3571)>*

*# Create a tf-idf dataframe*

*tfidf = pd.DataFrame(tf\_dtm.todense(),*

*……………………………columns=vectorizer.get\_feature\_names\_out(),*

*……………………………index=tweets.index)*

*tfidf.head()*

| ***aa*** | ***aadv*** | ***aadvantage*** | ***aal*** | ***abandon*** | ***abc*** | ***ability*** | ***able*** | ***aboard*** | ***abq*** | ***...*** | ***yummy*** | ***yup*** | ***yvonne*** | ***yvr*** | ***yyj*** | ***yyz*** | ***zero*** | ***zone*** | ***zoom*** | ***zurich*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *...* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* |
| *1* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *...* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* |
| *2* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *...* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* |
| *3* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *...* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* |
| *4* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *...* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* |

Quizás hayas notado que el tamaño del vocabulario es el mismo que vimos en el Desafío 2. Esto se debe a que usamos la misma configuración de parámetros al crear la vectorización. Sin embargo, los valores de la matriz son diferentes: son puntuaciones de **tf-idf** en lugar de conteos brutos.

**Interpretar valores de TF-IDF**

Analicemos el documento donde un término tiene los valores de **tf-idf** más altos. Usaremos el método **.idxmax()** para encontrar el índice.

*# Retrieve the index of the document*

*tfidf.idxmax()*

*aa 10077*

*aadv 9285*

*aadvantage 9974*

*aal 10630*

*abandon 7859*

*...*

*yyz 1350*

*zero 2705*

*zone 3177*

*zoom 3920*

*zurich 10622*

*Length: 3571, dtype: int64*

Por ejemplo, el término "**peor**" aparece de forma más clara en el tweet número 918th.

*tfidf.idxmax()['worst']*

*np.int64(918)*

¡Recordemos que este es el tweet donde la palabra “**peor**” aparece seis veces!

*tweets['text\_processed'].iloc[918]*

*"USER is the worst. worst reservation policies. worst costumer service. worst worst worst. congrats, USER you're not that bad!"*

¿Qué tal "**Cancelar**"? Veamos otro ejemplo.

*tfidf.idxmax()['cancel']*

*np.int64(5945)*

*tweets['text\_processed'].iloc[5945]*

*'USER cancelled flighted 😢'*

**🥊 Desafío 3: Palabras con las puntuaciones medias más altas en TF-IDF**

Hemos obtenido valores **tf-idf** para cada término en cada documento. Pero ¿qué nos dicen estos valores sobre el sentimiento de los **tweets**? ¿Hay palabras que sean especialmente informativas para los **tweets** positivos/negativos?

Para explorar esto, recopilemos los índices de todos los **tweets** positivos/negativos y calculemos la media de las puntuaciones **tf-idf** de las palabras que aparecen en cada categoría.

Hemos proporcionado el siguiente código de inicio como guía:

* Cree un subconjunto del **dataframe** de **tweets** según la etiqueta **airline\_sentiment** y recupere el índice de cada subconjunto **(.index**). Asigne el índice a **positive\_index** o a **negative\_index**.
* Para cada subconjunto:
* Recupere la representación **td-idf**
* Tome la media de los valores **tf-idf** del subconjunto con **.mean()**
* Ordene la media de los valores **tf-idf** en orden descendente con **.sort\_values()**
* Obtenga los 10 términos principales con **.head()**

A continuación, ejecute **pos.plot** y **neg.plot** para representar gráficamente las palabras con las puntuaciones medias **tf-idf** más altas para cada subconjunto.

*# Complete the boolean masks*

*positive\_index = tweets[tweets['airline\_sentiment'] == 'positive'].index*

*negative\_index = tweets[tweets['airline\_sentiment'] == 'negative'].index*

*# Complete the following two lines*

*pos = tfidf.loc[positive\_index].mean().sort\_values(ascending=False).head(10)*

*neg = tfidf.loc[negative\_index].mean().sort\_values(ascending=False).head(10)*

*pos.plot(kind='barh',*

*………….xlim=(0, 0.18),*

*………….color='cornflowerblue',*

*………….title='Top 10 terms with the highest mean tf-idf values for positive tweets');*

*Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.*

*neg.plot(kind='barh',*

*.………….xlim=(0, 0.18),*

*.………….color='darksalmon',*

*.………….title='Top 10 terms with the highest mean tf-idf values for negative tweets');*

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

🔔 Pregunta: ¿Cómo interpretarías estos resultados? ¡Comparte tu opinión en el chat!