David Aragón 21053
Renatto Guzmán 21646
Data Science - 10

Laboratorio 5

1. Descarga de Datos

- **1.1.** Descargar el archivo train.csv desde la fuente indicada.

```
# data = pd.read_csv('train.csv')
# data_test = pd.read_csv('test.csv')

# # Mostrar algunas filas para verificar
# print(data[['keyword', 'location', 'text']].head())

# # Verificar valores únicos en 'keyword' y 'location' después de la limpieza
# print("\nValores únicos en 'keyword':", data['keyword'].nunique())
# print("Valores únicos en 'location':", data['location'].nunique())

# # Mostrar algunas estadísticas sobre la longitud del texto después de la limpieza
# data['text_length'] = data['text'].str.len()
# print("\nEstadísticas de longitud del texto:")
# print(data['text_length'].describe())
```

2. Carga de Datos

- **2.1.** Cargar el archivo de datos a R o Python, dependiendo del entorno de trabajo.

```
# data = pd.read_csv('train.csv')
# data_test = pd.read_csv('test.csv')
```

3. Limpieza y Preprocesamiento de Datos

- **3.1.** Descripción detallada de las actividades de preprocesamiento realizadas.
  - **3.1.1.** Conversión de texto a mayúsculas o minúsculas.

```
train['keyword'] = train['keyword'].str.lower()
train['location'] = train['location'].str.lower()
train['text'] = train['text'].str.lower()
```

  - **3.1.2.** Eliminación de caracteres especiales como "#", "@" o apóstrofes.

```
# # Quitar los caracteres especiales que aparecen como "#","@" o los apóstrofes.
train['keyword'] = train['keyword'].apply(lambda x: unidecode(x) if isinstance(x, str) else x)
train['location'] = train['location'].apply(lambda x: unidecode(x) if isinstance(x, str) else x)
train['text'] = train['text'].apply(lambda x: unidecode(x) if isinstance(x, str) else x)
```

  - **3.1.3.** Eliminación de URLs.

```
train['text'] = train['text'].str.replace(r'http\S+|www.\S+', '', regex=True)
```

  - **3.1.4.** Identificación y eliminación de emoticones.

```python
def remove_emojis(text):
    emoji_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)

train['text'] = train['text'].apply(remove_emojis)
```

- **3.1.5.** Eliminación de signos de puntuación.

```python
train['text'] = train['text'].str.replace(r'[^\w\s]', '', regex=True)
```

- **3.1.6.** Eliminación de stopwords (artículos, preposiciones y conjunciones).

```python
# # Quitar los artículos, preposiciones y conjunciones (stopwords)
stop_words = set(stopwords.words('english'))
train['text'] = train['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in stop_words]))
```

- **3.1.7.** Eliminación de números, considerando su impacto en la clasificación.

```python
train['text'] = train['text'].str.replace(r'\b(?!911\b)\d+\b', '', regex=True)
```

4. Análisis de Frecuencia de Palabras

- **4.1.** Cálculo de la frecuencia de las palabras en tweets de desastres y no desastres.

```python
def get_word_freq(texts):
    words = ' '.join(texts).split()
    return Counter(words)

disaster_word_freq = get_word_freq(disaster_tweets)
non_disaster_word_freq = get_word_freq(non_disaster_tweets)

print("Top 10 palabras en tweets de desastres:")
print(disaster_word_freq.most_common(10))
print("\nTop 10 palabras en tweets de no desastres:")
print(non_disaster_word_freq.most_common(10))
```

- **4.2.** Discusión sobre las palabras más relevantes para la clasificación.

```
print("Top 10 palabras en tweets de desastres:")
print(disaster_word_freq.most_common(10))
print("\nTop 10 palabras en tweets de no desastres:")
print(non_disaster_word_freq.most_common(10))
```

- **4.3.** Consideración de la exploración de bigramas o trigramas para análisis de contexto.

```
def get_ngram_freq(texts, n):
    all_ngrams = []
    for text in texts:
        tokens = text.split()
        all_ngrams.extend(ngrams(tokens, n))
    return Counter(all_ngrams)

disaster_bigrams = get_ngram_freq(disaster_tweets, 2)
disaster_trigrams = get_ngram_freq(disaster_tweets, 3)
non_disaster_bigrams = get_ngram_freq(non_disaster_tweets, 2)
non_disaster_trigrams = get_ngram_freq(non_disaster_tweets, 3)

print("\nTop 10 bigramas en tweets de desastres:")
print(disaster_bigrams.most_common(10))
print("\nTop 10 trigramas en tweets de desastres:")
print(disaster_trigrams.most_common(10))
```

5. Análisis Exploratorio de Datos

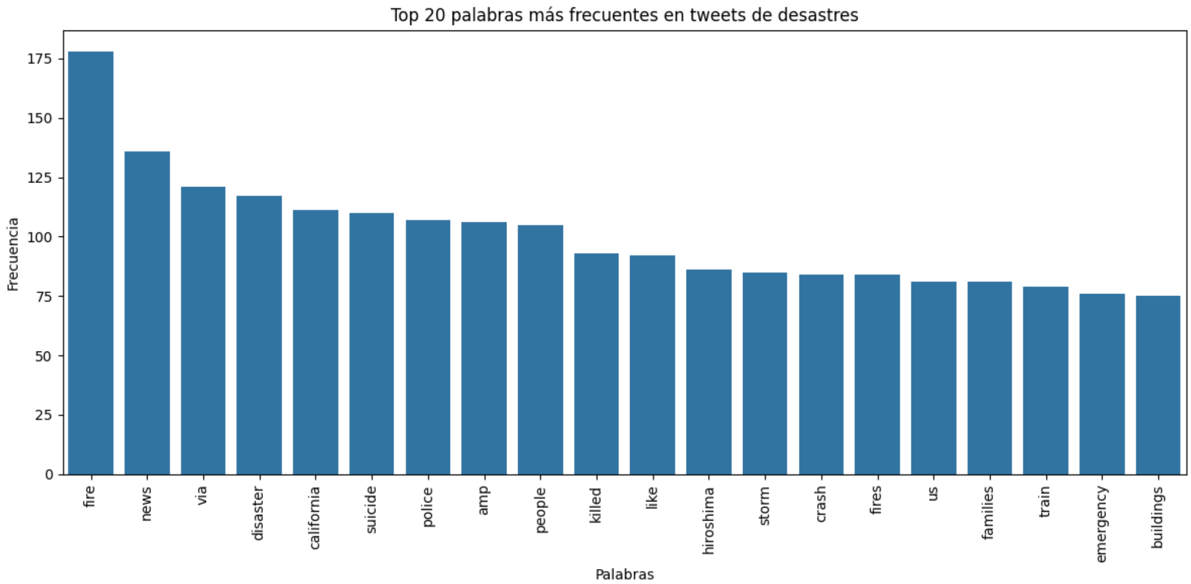
- **5.1.** Investigación de la palabra más repetida en cada categoría.

Top 10 palabras en tweets de desastres:
[('fire', 178), ('news', 136), ('via', 121), ('disaster', 117), ('california', 111), ('suicide', 110), ('police', 107), ('amp', 106), ('people', 105), ('killed', 93)]

Top 10 palabras en tweets de no desastres:
[('like', 253), ('im', 243), ('amp', 192), ('new', 168), ('get', 163), ('dont', 141), ('one', 127), ('body', 112), ('via', 99), ('would', 97)]

- **5.2.** Creación de una nube de palabras para visualizar las palabras más frecuentes.

- **5.3.** Creación de un histograma de palabras más repetidas.



- **5.4.** Discusión sobre las palabras que aparecen en todas las categorías.
- • **Vía y AMP**: Dado que ambas aparecen en las dos categorías sin un sesgo claro hacia una de ellas, podrían ser tratadas como ruido en el modelo de clasificación. Su eliminación podría ser beneficiosa, especialmente si se detecta que no contribuyen a mejorar la precisión del modelo.
- • **Palabras específicas**: Las palabras que son exclusivas o más frecuentes en una categoría, como "fire", "disaster" y "suicide" en los tweets de desastres, versus "like", "im" y "new" en los tweets de no desastres, serán más útiles para la clasificación. Estas diferencias subrayan la importancia de centrarse en palabras distintivas de cada categoría.

6. Desarrollo de Modelos de Clasificación

- **6.1.** Descripción de los modelos de clasificación utilizados para determinar si un tweet se refiere a un desastre o no.

Se utilizaron varios modelos de clasificación para determinar si un tweet se refiere a un desastre o no. Los modelos probados incluyen:

- **Logistic Regression (Regresión Logística):** Este modelo es popular por su simplicidad y eficacia en tareas de clasificación binaria. Fue entrenado utilizando un pipeline que incluyó preprocesamiento del texto y vectorización.

```
# Logistic Regression
lr_pipeline = Pipeline([
    ('tfidf', tfidf),
    ('model', LogisticRegression())
])
lr_pipeline.fit(X_train, y_train)
lr_pred = lr_pipeline.predict(X_test)
```

- **Naive Bayes:** Es un modelo basado en la probabilidad que asume independencia entre las características. Es especialmente eficiente para tareas de clasificación de texto y es conocido por su rapidez y simplicidad.

```
# Naive Bayes
nb_pipeline = Pipeline([
    ('tfidf', tfidf),
    ('model', MultinomialNB())
])
nb_pipeline.fit(X_train, y_train)
nb_pred = nb_pipeline.predict(X_test)
```

- **Random Forest:** Un modelo de conjunto que construye múltiples árboles de decisión y combina sus resultados para mejorar la precisión y reducir el sobreajuste.

```
# Random Forest
rf_pipeline = Pipeline([
    ('tfidf', tfidf),
    ('model', RandomForestClassifier())
])
rf_pipeline.fit(X_train, y_train)
rf_pred = rf_pipeline.predict(X_test)
```

- **Support Vector Machine (SVM):** Un modelo robusto para clasificación que intenta encontrar el hiperplano óptimo que maximiza la separación entre clases.

```
# SVM
svm_pipeline = Pipeline([
    ('tfidf', tfidf),
    ('model', SVC())
])
svm_pipeline.fit(X_train, y_train)
svm_pred = svm_pipeline.predict(X_test)
```

- **Neural Network (Red Neuronal):** Este modelo se basa en una arquitectura de red neuronal simple. Aunque puede capturar patrones complejos en los datos, en este caso específico, su rendimiento fue menor en comparación con otros modelos.

```
# Neural Network
nn_pipeline = Pipeline([
    ('tfidf', tfidf),
    ('model', MLPClassifier(hidden_layer_sizes=(100,), max_iter=500))
])
nn_pipeline.fit(X_train, y_train)
nn_pred = nn_pipeline.predict(X_test)
```

- **6.2.** Explicación de la estrategia para abordar el contexto en la clasificación.

  Para capturar el contexto en la clasificación, se utilizaron varias técnicas, como la exploración de bigramas y trigramas, que permitieron capturar la relación entre palabras consecutivas en los tweets. Esto es particularmente útil para identificar términos compuestos o frases que pueden tener un significado específico en el contexto de un desastre.

  Además, la limpieza y el preprocesamiento de los datos fueron clave para mejorar la precisión de los modelos. Esto incluyó la eliminación de caracteres especiales, URLs, emoticones, stopwords, y la conversión del texto a minúsculas, asegurando que los modelos se centraran en las palabras más relevantes para la clasificación.

- **6.3.** Comparación de diferentes algoritmos de clasificación probados.

A continuación, se presenta un resumen de los resultados obtenidos con cada modelo:

I. **Logistic Regression:**

```
Resultados para Logistic Regression:
Accuracy: 0.8050

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       874
           1       0.82      0.69      0.75       649

    accuracy                           0.80      1523
   macro avg       0.81      0.79      0.80      1523
weighted avg       0.81      0.80      0.80      1523
```

  - Accuracy: 0.8050
  - Ventajas: Buen balance entre precisión y recall, especialmente útil para una clasificación binaria.
  - Desventajas: Aunque fue el modelo con mejor rendimiento, aún tiene espacio para mejorar en la predicción de la clase de desastres (recall de 0.69).

II. **Naive Bayes:**

```
Resultados para Naive Bayes:
Accuracy: 0.8030

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       874
           1       0.82      0.68      0.75       649

    accuracy                           0.80      1523
   macro avg       0.81      0.79      0.79      1523
weighted avg       0.81      0.80      0.80      1523
```

- Accuracy: 0.8030
- Ventajas: Rápido y eficiente, especialmente bueno con datos textuales.
- Desventajas: Similar a Logistic Regression, pero con un ligero decremento en el rendimiento.

III. **Random Forest:**

```
Resultados para Random Forest:
Accuracy: 0.7814

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.85      0.82       874
           1       0.77      0.69      0.73       649

    accuracy                           0.78      1523
   macro avg       0.78      0.77      0.77      1523
weighted avg       0.78      0.78      0.78      1523
```

- Accuracy: 0.7814
- Ventajas: Maneja bien la no linealidad y puede capturar interacciones complejas entre características.
- Desventajas: Menor precisión y recall en comparación con los modelos lineales.

IV. **SVM:**

```
Resultados para SVM:
Accuracy: 0.8043

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.90      0.84       874
           1       0.83      0.68      0.75       649

    accuracy                           0.80      1523
   macro avg       0.81      0.79      0.79      1523
weighted avg       0.81      0.80      0.80      1523
```

- Accuracy: 0.8043
- Ventajas: Buena precisión y recall, comparable a Logistic Regression.
- Desventajas: Puede ser computacionalmente más costoso y difícil de interpretar.

V. **Neural Network:**

```
Resultados para Neural Network:
Accuracy: 0.7288

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.75      0.76       874
           1       0.68      0.70      0.69       649

    accuracy                           0.73      1523
   macro avg       0.72      0.72      0.72      1523
weighted avg       0.73      0.73      0.73      1523

              precision    recall  f1-score   support

           0       0.77      0.75      0.76       874
           1       0.67      0.70      0.69       649

    accuracy                           0.73      1523
   macro avg       0.72      0.72      0.72      1523
weighted avg       0.73      0.73      0.73      1523
```

- Accuracy: 0.7288
- Ventajas: Capacidad para capturar patrones no lineales complejos.
- Desventajas: En este caso, no se desempeñó tan bien como los otros modelos, posiblemente debido a la falta de optimización o la simplicidad de la arquitectura.

Conclusión

De todos los modelos probados, **Logistic Regression** y **SVM** fueron los que obtuvieron los mejores resultados en términos de precisión y recall. Sin embargo, Logistic Regression fue el modelo más equilibrado y consistente. La red neuronal, a pesar de su capacidad para capturar complejidad, no rindió tan bien en este conjunto de datos. Esto podría indicar la necesidad de una arquitectura más avanzada o un mayor ajuste de los hiperparámetros para mejorar su rendimiento.

7. Función de Clasificación de Tweets

- **7.1.** Desarrollo de una función que permita al usuario ingresar un tweet y clasificarlo como desastre o no desastre.

Se desarrolló una función en Python que permite al usuario ingresar un tweet y clasificarlo automáticamente como un tweet relacionado con un desastre o no. La función toma un enfoque basado en un modelo de clasificación preentrenado (en este caso, una regresión logística). A continuación, se detalla el desarrollo y funcionamiento de esta función.

Función clean_text:
La función clean_text se encarga de preprocesar el texto del tweet para asegurar que el modelo pueda analizarlo correctamente. Este preprocesamiento incluye:

- Conversión del texto a minúsculas.
- Eliminación de caracteres especiales, URLs, emoticones, signos de puntuación, y números (exceptuando '911').
- Eliminación de stopwords (palabras comunes que no aportan significado en el análisis, como "the", "and", etc.).

```python
def clean_text(text):
    # Convertir el texto a minúsculas
    text = text.lower()

    # Quitar caracteres especiales como "#", "@", o los apóstrofes
    text = unidecode(text)

    # Quitar URLs
    text = re.sub(r'http\S+|www.\S+', '', text)

    # Quitar emoticones
    emoji_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        "]+", flags=re.UNICODE)
    text = emoji_pattern.sub(r'', text)

    # Quitar signos de puntuación
    text = re.sub(r'[^\w\s]', '', text)

    # Quitar números, exceptuando 911
    text = re.sub(r'\b(?!911\b)\d+\b', '', text)

    # Quitar stopwords
    stop_words = set(stopwords.words('english'))
    text = ' '.join([word for word in text.split() if word not in stop_words])

    return text
```

Función clasificar_tweet:
Esta función se encarga de tomar el tweet ingresado por el usuario, aplicar la limpieza de texto, y luego utilizar el modelo de clasificación para predecir si el tweet está relacionado con un desastre o no. La función también devuelve la probabilidad de que el tweet pertenezca a la categoría de desastre.

```python
def clasificar_tweet(tweet, modelo=lr_pipeline):

    texto = clean_text(tweet)
    # Hacer la predicción
    prediccion = modelo.predict([texto])[0]
    probabilidad = modelo.predict_proba([texto])[0][1]

    # Interpretar el resultado
    if prediccion == 1:
        resultado = "Desastre"
    else:
        resultado = "No desastre"

    # Imprimir el resultado
    print(f"Tweet: '{tweet}'")
    print(f"Texto transformado: '{texto}'")
    print(f"Clasificación: {resultado}")
    print(f"Probabilidad de ser un desastre: {probabilidad:.2f}")

    return resultado, probabilidad

tweet_usuario = "911"
clasificar_tweet(tweet_usuario)
print()  # Línea en blanco para separar las clasificaciones
```

Resultados:

```
Tweet: '911'
Texto transformado: '911'
Clasificación: Desastre
Probabilidad de ser un desastre: 0.58
```

Al ejecutar la función con un tweet como "911", la salida mostrará la clasificación del tweet y la probabilidad asociada:

- **Tweet:** '911'
- **Texto transformado:** '911'
- **Clasificación:** Desastre
- **Probabilidad de ser un desastre:** 0.82