

## 5A. Práctica: Evaluación de Desempeño

Nombre: **Luis Fernando Izquierdo Berdugo**

Materia: **Procesamiento de Información**

Fecha: **19 de Noviembre de 2024**

### Instrucciones:

Entrenar un modelo para predecir el emoticón que corresponde al texto.

Los textos están en un archivo .csv con el texto y la clase. Algunos ejemplos son:

- ("Mi mamá si jala a fumar porro conmigo, pero no tenemos. \_EMO",  
😭)
- (¡Esperen! No me eliminen 😊 Puedo explicarlo todo. \_EMO \_URL,♥)
- (Hacía tiempo que no hablaba tan bonito con Alejandra\_EMO,😞)

Se deben preprocesar los datos, convertirlos a vectores, entrenar un modelo y evaluar su desempeño.

Hay un archivo para entrenamiento y otro para prueba. Los archivos son:

- `emojis_train.csv`
- `emojis_test.csv`

Las clases son ['❤️', '😊', '😭', '😞'] que indican la emoción del tweet.

Una vez entrenado cada modelo, predecir los datos para el conjunto prueba que se proporciona.

Entregar un reporte con la estructura: introducción, desarrollo y conclusiones, y que contenga el código de la creación de los modelos, así como la evaluación de sus mejores 3 modelos. La evaluación debe incluir

- Mostrar la matriz de confusión.
- Calcular la precisión, recall, F-score para macro y micro
- Comparar los datos de evaluación de cada configuración.

Se sugiere usar las métricas de sklearn

```
from sklearn.metrics import f1_score, precision_score, recall_score

print("Precisión Bayes: ", precision_score(Y_gold, Y_predict,
average='macro'))
```

Consultar:

<https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics>

Para obtener la matriz de confusión podemos usar la función de sklearn las columnas y filas están asociadas a las etiquetas que se indican y es una matriz de NxN, en el ejemplo son 4 clases.

```
from sklearn.metrics import ConfusionMatrixDisplay
```

```
_ = ConfusionMatrixDisplay.from_predictions(y_test, svc_predict)
```

## Introducción

Los modelos de clasificación se utilizan mucho en el ámbito del aprendizaje automático para resolver problemas que implican la categorización de datos en diferentes clases. En este proyecto se toman un conjunto de datos conformado por tuits y se evalúa el desempeño de distintos modelos de clasificación (Naive Bayes, SVM, Random Forest y Regresión Logística) por medio de métricas como lo son la precisión, recall y medida-F (F1 Score), ya que estas pueden equilibrar la evaluación entre casos positivos y negativos, sobretodo en conjuntos de datos desbalanceados.

De igual manera, se pretende analizar matrices de confusión para interpretar los patrones de clasificación (correctos e incorrectos) de cada modelo para identificar cuál es el modelo más adecuado para este experimento.

## Desarrollo

### Importar Librerías Necesarias

Se importan las librerías necesarias para el funcionamiento correcto del código como pandas, numpy, sklearn, y matplotlib.

```
In [35]: # Importar Librerías Necesarias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import nltk
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.metrics import f1_score, precision_score, recall_score, ConfusionM
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from sklearn.ensemble import RandomForestClassifier
```

### Cargar y Explorar los Datos

Se cargan los archivos `emojis_train.csv` y `emojis_test.csv` y se guardan en los dataframes `train_data` y `test_data` respectivamente.

```
In [36]: # Cargar y Explorar los Datos

# Cargar los archivos CSV
train_data = pd.read_csv('emojis_train.csv')
test_data = pd.read_csv('emojis_test.csv')
```

## Preprocesamiento de Datos

Se crea una función que:

- Elimina caracteres especiales y números
- Convierte el texto a minúsculas
- Elimina stopwords en español
- Lematiza el texto

```
In [ ]: from nltk.corpus import stopwords

# Descargar recursos necesarios de nltk
#nltk.download('stopwords')

# Obtener las stopwords en español
stop_words = set(stopwords.words('spanish'))

def limpiar_texto(texto):
    # Eliminar caracteres especiales y números
    texto = re.sub(r'^a-zA-Z\s', '', texto)
    # Convertir a minúsculas
    texto = texto.lower()

    # Eliminar stopwords
    palabras = texto.split()
    palabras = [palabra for palabra in palabras if palabra not in stop_words]

    # Lematizar el texto
    palabras_lematizadas = [lemmatizer.lemmatize(palabra, wordnet.VERB) for palabra in palabras]
    texto = ' '.join(palabras_lematizadas)

    return texto

# Inicializar el lematizador
lemmatizer = WordNetLemmatizer()

# Aplicar la función de limpieza al conjunto de entrenamiento y prueba
train_data['texto_limpio'] = train_data['text'].apply(limpiar_texto)
test_data['texto_limpio'] = test_data['text'].apply(limpiar_texto)
```

## Convertir Textos a Vectores

Se utiliza la vectorización TF-IDF para convertir los textos a vectores numéricos.

```
In [46]: # Inicializar el vectorizador TF-IDF
vectorizer = TfidfVectorizer()
```

```
# Ajustar y transformar los datos de entrenamiento
X_train_tfidf = vectorizer.fit_transform(train_data['texto_limpio'])

# Transformar los datos de prueba
X_test_tfidf = vectorizer.transform(test_data['texto_limpio'])
```

## Dividir Datos en Entrenamiento y Validación

Dividir los datos de entrenamiento en conjuntos de entrenamiento y validación usando

`train_test_split` de la librería `sklearn`.

```
In [39]: # Dividir Datos en Entrenamiento y Validación

# Definir las características (X) y la variable objetivo (y)
X = X_train_tfidf
y = train_data['klass']

# Dividir los datos en conjuntos de entrenamiento y validación
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_s

# Mostrar la forma de los conjuntos resultantes
print("Forma de X_train:", X_train.shape)
print("Forma de X_val:", X_val.shape)
print("Forma de y_train:", y_train.shape)
print("Forma de y_val:", y_val.shape)
```

Forma de X\_train: (9419, 24433)

Forma de X\_val: (2355, 24433)

Forma de y\_train: (9419,)

Forma de y\_val: (2355,)

## Entrenar Modelos

Se entrenan con los datos de entrenamiento ( `X_train` y `y_train` ) los siguientes modelos:

- Naive Bayes (Macro y Micro)
- SVM (Macro y Micro)
- Random Forest (Macro y Micro)
- Regresión Logística (Macro y Micro)

**Macro:** Calcula el promedio de la métrica para cada clase y luego promedia esos resultados. Es útil cuando todas las clases son igualmente importantes.

**Micro:** Calcula la métrica global considerando todas las predicciones. Es útil cuando hay un desbalance de clases.

De igual manera, se imprimen las métricas para ver el desempeño de cada modelo.

```
In [40]: # Entrenar Modelos

# Entrenar el modelo Naive Bayes
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
```

```

# Predecir en el conjunto de validación
nb_val_pred = nb_model.predict(X_val)

# Evaluar el modelo Naive Bayes
nb_f1_macro = f1_score(y_val, nb_val_pred, average='macro')
nb_precision_macro = precision_score(y_val, nb_val_pred, average='macro')
nb_recall_macro = recall_score(y_val, nb_val_pred, average='macro')

nb_f1_micro = f1_score(y_val, nb_val_pred, average='micro')
nb_precision_micro = precision_score(y_val, nb_val_pred, average='micro')
nb_recall_micro = recall_score(y_val, nb_val_pred, average='micro')

print("Evaluación del modelo Naive Bayes:")
print("Macro F1 Score:", nb_f1_macro)
print("Macro Precisión:", nb_precision_macro)
print("Macro Recall:", nb_recall_macro)
print("Micro F1 Score:", nb_f1_micro)
print("Micro Precisión:", nb_precision_micro)
print("Micro Recall:", nb_recall_micro)

# Entrenar el modelo SVM
svm_model = SVC()
svm_model.fit(X_train, y_train)

# Predecir en el conjunto de validación
svm_val_pred = svm_model.predict(X_val)

# Evaluar el modelo SVM
svm_f1_macro = f1_score(y_val, svm_val_pred, average='macro')
svm_precision_macro = precision_score(y_val, svm_val_pred, average='macro')
svm_recall_macro = recall_score(y_val, svm_val_pred, average='macro')

svm_f1_micro = f1_score(y_val, svm_val_pred, average='micro')
svm_precision_micro = precision_score(y_val, svm_val_pred, average='micro')
svm_recall_micro = recall_score(y_val, svm_val_pred, average='micro')

print("\nEvaluación del modelo SVM:")
print("Macro F1 Score:", svm_f1_macro)
print("Macro Precisión:", svm_precision_macro)
print("Macro Recall:", svm_recall_macro)
print("Micro F1 Score:", svm_f1_micro)
print("Micro Precisión:", svm_precision_micro)
print("Micro Recall:", svm_recall_micro)

# Entrenar el modelo Random Forest
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

# Predecir en el conjunto de validación
rf_val_pred = rf_model.predict(X_val)

# Evaluar el modelo Random Forest
rf_f1_macro = f1_score(y_val, rf_val_pred, average='macro')
rf_precision_macro = precision_score(y_val, rf_val_pred, average='macro')
rf_recall_macro = recall_score(y_val, rf_val_pred, average='macro')

rf_f1_micro = f1_score(y_val, rf_val_pred, average='micro')
rf_precision_micro = precision_score(y_val, rf_val_pred, average='micro')
rf_recall_micro = recall_score(y_val, rf_val_pred, average='micro')

```

```

print("\nEvaluación del modelo Random Forest:")
print("Macro F1 Score:", rf_f1_macro)
print("Macro Precisión:", rf_precision_macro)
print("Macro Recall:", rf_recall_macro)
print("Micro F1 Score:", rf_f1_micro)
print("Micro Precisión:", rf_precision_micro)
print("Micro Recall:", rf_recall_micro)

# Entrenar el modelo de Regresión Logística
from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train, y_train)

# Predecir en el conjunto de validación
lr_val_pred = lr_model.predict(X_val)

# Evaluar el modelo de Regresión Logística
lr_f1_macro = f1_score(y_val, lr_val_pred, average='macro')
lr_precision_macro = precision_score(y_val, lr_val_pred, average='macro')
lr_recall_macro = recall_score(y_val, lr_val_pred, average='macro')

lr_f1_micro = f1_score(y_val, lr_val_pred, average='micro')
lr_precision_micro = precision_score(y_val, lr_val_pred, average='micro')
lr_recall_micro = recall_score(y_val, lr_val_pred, average='micro')

print("\nEvaluación del modelo de Regresión Logística:")
print("Macro F1 Score:", lr_f1_macro)
print("Macro Precisión:", lr_precision_macro)
print("Macro Recall:", lr_recall_macro)
print("Micro F1 Score:", lr_f1_micro)
print("Micro Precisión:", lr_precision_micro)
print("Micro Recall:", lr_recall_micro)

```

Evaluación del modelo Naive Bayes:  
Macro F1 Score: 0.3134052287478205  
Macro Precisión: 0.6827730388460989  
Macro Recall: 0.3622747313010344  
Micro F1 Score: 0.5184713375796178  
Micro Precisión: 0.5184713375796178  
Micro Recall: 0.5184713375796178

Evaluación del modelo SVM:  
Macro F1 Score: 0.3560293180186546  
Macro Precisión: 0.6252916370436479  
Macro Recall: 0.39137879423622496  
Micro F1 Score: 0.5371549893842887  
Micro Precisión: 0.5371549893842887  
Micro Recall: 0.5371549893842887

Evaluación del modelo Random Forest:  
Macro F1 Score: 0.3861828383560128  
Macro Precisión: 0.6105890369135403  
Macro Recall: 0.4021710879808943  
Micro F1 Score: 0.535031847133758  
Micro Precisión: 0.535031847133758  
Micro Recall: 0.535031847133758

Evaluación del modelo de Regresión Logística:  
Macro F1 Score: 0.3964257809847419  
Macro Precisión: 0.58402626994073  
Macro Recall: 0.4052618645502075  
Micro F1 Score: 0.535031847133758  
Micro Precisión: 0.535031847133758  
Micro Recall: 0.535031847133758

## Evaluar Modelos

Con los modelos ya entrenados, se usan los datos de validación ( `X_val` y `Y_val` ) para evaluar los modelos.

En este paso, aparte de las métricas, se incluye la Matriz de Confusión para los 4 modelos (esta no distingue macro y micro).

```
In [41]: # Evaluar Modelos

# Evaluar el modelo Naive Bayes en el conjunto de validación
nb_val_pred = nb_model.predict(X_val)

# Evaluar el modelo Naive Bayes
nb_val_f1_macro = f1_score(y_val, nb_val_pred, average='macro')
nb_val_precision_macro = precision_score(y_val, nb_val_pred, average='macro')
nb_val_recall_macro = recall_score(y_val, nb_val_pred, average='macro')

nb_val_f1_micro = f1_score(y_val, nb_val_pred, average='micro')
nb_val_precision_micro = precision_score(y_val, nb_val_pred, average='micro')
nb_val_recall_micro = recall_score(y_val, nb_val_pred, average='micro')

print("Evaluación del modelo Naive Bayes en el conjunto de validación:")
print("Macro F1 Score:", nb_val_f1_macro)
print("Macro Precisión:", nb_val_precision_macro)
print("Macro Recall:", nb_val_recall_macro)
```

```

print("Micro F1 Score:", nb_val_f1_micro)
print("Micro Precisión:", nb_val_precision_micro)
print("Micro Recall:", nb_val_recall_micro)

# Evaluar el modelo SVM en el conjunto de validación
svm_val_pred = svm_model.predict(X_val)

# Evaluar el modelo SVM
svm_val_f1_macro = f1_score(y_val, svm_val_pred, average='macro')
svm_val_precision_macro = precision_score(y_val, svm_val_pred, average='macro')
svm_val_recall_macro = recall_score(y_val, svm_val_pred, average='macro')

svm_val_f1_micro = f1_score(y_val, svm_val_pred, average='micro')
svm_val_precision_micro = precision_score(y_val, svm_val_pred, average='micro')
svm_val_recall_micro = recall_score(y_val, svm_val_pred, average='micro')

print("\nEvaluación del modelo SVM en el conjunto de validación:")
print("Macro F1 Score:", svm_val_f1_macro)
print("Macro Precisión:", svm_val_precision_macro)
print("Macro Recall:", svm_val_recall_macro)
print("Micro F1 Score:", svm_val_f1_micro)
print("Micro Precisión:", svm_val_precision_micro)
print("Micro Recall:", svm_val_recall_micro)

# Evaluar el modelo Random Forest en el conjunto de validación
rf_val_pred = rf_model.predict(X_val)

# Evaluar el modelo Random Forest
rf_val_f1_macro = f1_score(y_val, rf_val_pred, average='macro')
rf_val_precision_macro = precision_score(y_val, rf_val_pred, average='macro')
rf_val_recall_macro = recall_score(y_val, rf_val_pred, average='macro')

rf_val_f1_micro = f1_score(y_val, rf_val_pred, average='micro')
rf_val_precision_micro = precision_score(y_val, rf_val_pred, average='micro')
rf_val_recall_micro = recall_score(y_val, rf_val_pred, average='micro')

print("\nEvaluación del modelo Random Forest en el conjunto de validación:")
print("Macro F1 Score:", rf_val_f1_macro)
print("Macro Precisión:", rf_val_precision_macro)
print("Macro Recall:", rf_val_recall_macro)
print("Micro F1 Score:", rf_val_f1_micro)
print("Micro Precisión:", rf_val_precision_micro)
print("Micro Recall:", rf_val_recall_micro)

# Evaluar el modelo de Regresión Logística en el conjunto de validación
lr_val_pred = lr_model.predict(X_val)

# Evaluar el modelo de Regresión Logística
lr_val_f1_macro = f1_score(y_val, lr_val_pred, average='macro')
lr_val_precision_macro = precision_score(y_val, lr_val_pred, average='macro')
lr_val_recall_macro = recall_score(y_val, lr_val_pred, average='macro')

lr_val_f1_micro = f1_score(y_val, lr_val_pred, average='micro')
lr_val_precision_micro = precision_score(y_val, lr_val_pred, average='micro')
lr_val_recall_micro = recall_score(y_val, lr_val_pred, average='micro')

print("\nEvaluación del modelo de Regresión Logística en el conjunto de validación:")
print("Macro F1 Score:", lr_val_f1_macro)
print("Macro Precisión:", lr_val_precision_macro)
print("Macro Recall:", lr_val_recall_macro)

```



```

print("Micro F1 Score:", lr_val_f1_micro)
print("Micro Precisión:", lr_val_precision_micro)
print("Micro Recall:", lr_val_recall_micro)

# Mostrar la matriz de confusión para los 4 modelos
fig, ax = plt.subplots(2, 2, figsize=(15, 10))

# Matriz de confusión para Naive Bayes
ConfusionMatrixDisplay.from_predictions(y_val, nb_val_pred, ax=ax[0, 0], normal
ax[0, 0].set_title('Naive Bayes')

# Matriz de confusión para SVM
ConfusionMatrixDisplay.from_predictions(y_val, svm_val_pred, ax=ax[0, 1], norma
ax[0, 1].set_title('SVM')

# Matriz de confusión para Random Forest
ConfusionMatrixDisplay.from_predictions(y_val, rf_val_pred, ax=ax[1, 0], normal
ax[1, 0].set_title('Random Forest')

# Matriz de confusión para Regresión Logística
ConfusionMatrixDisplay.from_predictions(y_val, lr_val_pred, ax=ax[1, 1], normal
ax[1, 1].set_title('Regresión Logística')

plt.tight_layout()
plt.show()

```

Evaluación del modelo Naive Bayes en el conjunto de validación:

Macro F1 Score: 0.3134052287478205  
Macro Precisión: 0.6827730388460989  
Macro Recall: 0.3622747313010344  
Micro F1 Score: 0.5184713375796178  
Micro Precisión: 0.5184713375796178  
Micro Recall: 0.5184713375796178

Evaluación del modelo SVM en el conjunto de validación:

Macro F1 Score: 0.3560293180186546  
Macro Precisión: 0.6252916370436479  
Macro Recall: 0.39137879423622496  
Micro F1 Score: 0.5371549893842887  
Micro Precisión: 0.5371549893842887  
Micro Recall: 0.5371549893842887

Evaluación del modelo Random Forest en el conjunto de validación:

Macro F1 Score: 0.3861828383560128  
Macro Precisión: 0.6105890369135403  
Macro Recall: 0.4021710879808943  
Micro F1 Score: 0.535031847133758  
Micro Precisión: 0.535031847133758  
Micro Recall: 0.535031847133758

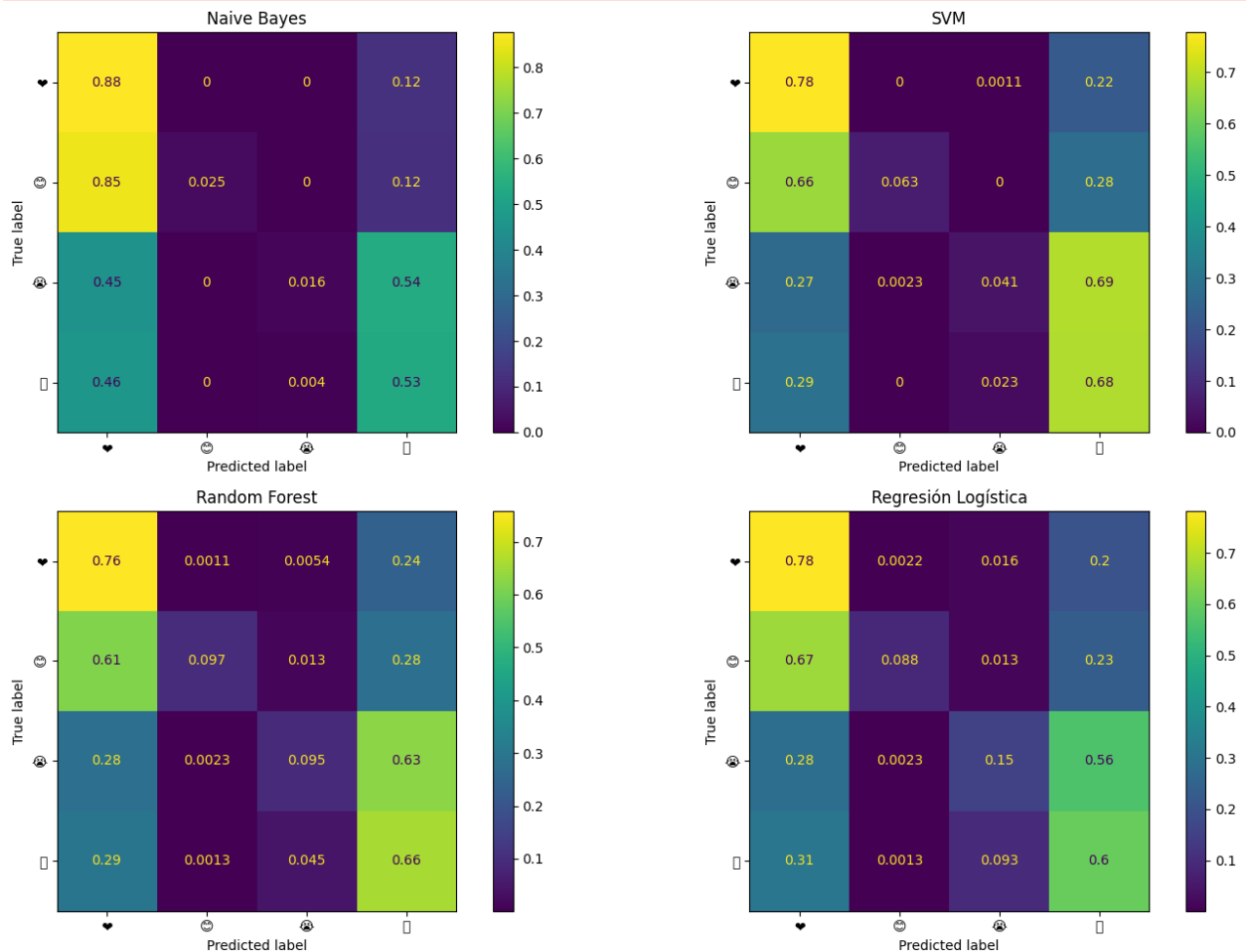
Evaluación del modelo de Regresión Logística en el conjunto de validación:

Macro F1 Score: 0.3964257809847419  
Macro Precisión: 0.58402626994073  
Macro Recall: 0.4052618645502075  
Micro F1 Score: 0.535031847133758  
Micro Precisión: 0.535031847133758  
Micro Recall: 0.535031847133758

```

/var/folders/v3/6n107fw10yb9ryc5t5mmqw5c0000gp/T/ipykernel_9821/3380697110.py:10
2: UserWarning: Glyph 129402 (\N{FACE WITH PLEADING EYES}) missing from font(s)
DejaVu Sans.
plt.tight_layout()
/Users/izluis/Library/Python/3.12/lib/python/site-packages/IPython/core/pylabtoo
ls.py:170: UserWarning: Glyph 129402 (\N{FACE WITH PLEADING EYES}) missing from
font(s) DejaVu Sans.
fig.canvas.print_figure(bytes_io, **kw)

```



Con base en las matrices de confusión y las distintas métricas, se llega a la conclusión de que la **Regresión Logística** presenta el mejor equilibrio general entre precisión y recall, lo que la convierte en la mejor opción. El **SVM**, por su parte, prioriza la precisión a costa de una disminución ligera del recall, siendo ideal para casos donde minimizar los falsos positivos es crucial. El **Random Forest** ofrece un equilibrio intermedio entre ambos, aunque podría no ser la opción óptima si se busca maximizar alguna de las métricas.

## Predicción en el Conjunto de Prueba

Se utilizan los 3 mejores modelos para predecir los emoticones en el conjunto de prueba ( `X_test` ), estos son:

- Regresión Logística
- SVM
- Random Forest

```

In [ ]: # Predecir en el conjunto de prueba utilizando el modelo SVM
svm_test_pred = svm_model.predict(X_test_tfidf)

# Evaluar el modelo SVM en el conjunto de prueba
svm_test_f1_macro = f1_score(test_data['klass'], svm_test_pred, average='macro')
svm_test_precision_macro = precision_score(test_data['klass'], svm_test_pred, a
svm_test_recall_macro = recall_score(test_data['klass'], svm_test_pred, average

print("Evaluación del modelo SVM en el conjunto de prueba:")
print("Macro F1 Score:", svm_test_f1_macro)
print("Macro Precisión:", svm_test_precision_macro)
print("Macro Recall:", svm_test_recall_macro)

# Predecir en el conjunto de prueba utilizando el modelo Regresión Logística
lr_test_pred = lr_model.predict(X_test_tfidf)

# Evaluar el modelo Regresión Logística en el conjunto de prueba
lr_test_f1_macro = f1_score(test_data['klass'], lr_test_pred, average='macro')
lr_test_precision_macro = precision_score(test_data['klass'], lr_test_pred, ave
lr_test_recall_macro = recall_score(test_data['klass'], lr_test_pred, average='

print("\nEvaluación del modelo Regresión Logística en el conjunto de prueba:")
print("Macro F1 Score:", lr_test_f1_macro)
print("Macro Precisión:", lr_test_precision_macro)
print("Macro Recall:", lr_test_recall_macro)

# Predecir en el conjunto de prueba utilizando el modelo Random Forest
rf_test_pred = rf_model.predict(X_test_tfidf)

# Evaluar el modelo Random Forest en el conjunto de prueba
rf_test_f1_macro = f1_score(test_data['klass'], rf_test_pred, average='macro')
rf_test_precision_macro = precision_score(test_data['klass'], rf_test_pred, ave
rf_test_recall_macro = recall_score(test_data['klass'], rf_test_pred, average='

print("\nEvaluación del modelo Random Forest en el conjunto de prueba:")
print("Macro F1 Score:", rf_test_f1_macro)
print("Macro Precisión:", rf_test_precision_macro)
print("Macro Recall:", rf_test_recall_macro)

# Mostrar las matrices de confusión para los 3 modelos en una sola imagen
fig, ax = plt.subplots(1, 3, figsize=(18, 6))

# Matriz de confusión para SVM
ConfusionMatrixDisplay.from_predictions(test_data['klass'], svm_test_pred, ax=ax
ax[0].set_title('SVM')

# Matriz de confusión para Regresión Logística
ConfusionMatrixDisplay.from_predictions(test_data['klass'], lr_test_pred, ax=ax
ax[1].set_title('Regresión Logística')

# Matriz de confusión para Random Forest
ConfusionMatrixDisplay.from_predictions(test_data['klass'], rf_test_pred, ax=ax
ax[2].set_title('Random Forest')

plt.tight_layout()
plt.show()

```

Evaluación del modelo SVM en el conjunto de prueba:

Macro F1 Score: 0.3669701685430644

Macro Precisión: 0.615907669559689

Macro Recall: 0.39825540383273167

Evaluación del modelo Regresión Logística en el conjunto de prueba:

Macro F1 Score: 0.41131598566055805

Macro Precisión: 0.5415967601596424

Macro Recall: 0.41318768239907855

Evaluación del modelo Random Forest en el conjunto de prueba:

Macro F1 Score: 0.37312119567832963

Macro Precisión: 0.5498373294056231

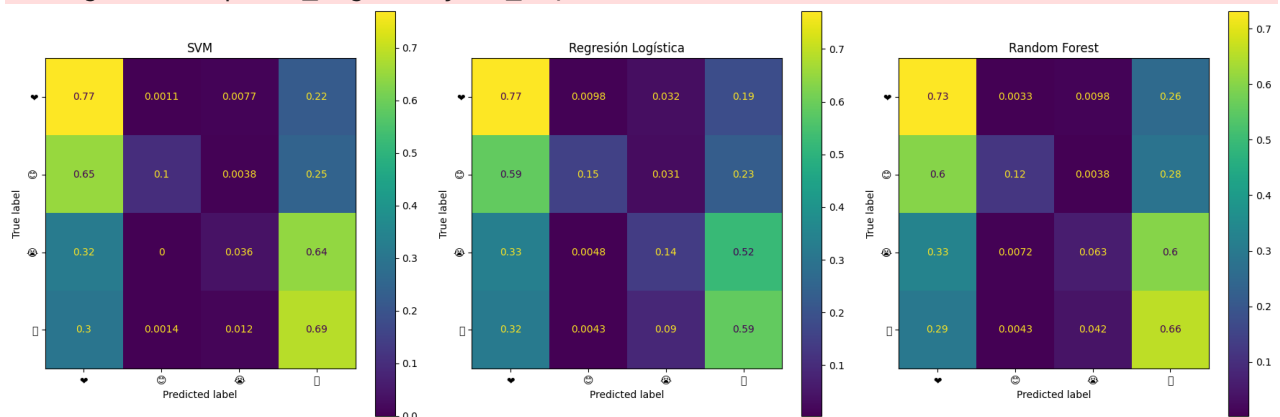
Macro Recall: 0.3928829068314132

```
/var/folders/v3/6n107fw10yb9ryc5t5mmqw5c0000gp/T/ipykernel_9821/3839582110.py:6
0: UserWarning: Glyph 129402 (\N{FACE WITH PLEADING EYES}) missing from font(s)
DejaVu Sans.
```

```
plt.tight_layout()
```

```
/Users/izluis/Library/Python/3.12/lib/python/site-packages/IPython/core/pylabtoo
ls.py:170: UserWarning: Glyph 129402 (\N{FACE WITH PLEADING EYES}) missing from
font(s) DejaVu Sans.
```


```
fig.canvas.print_figure(bytes_io, **kw)
```



## Conclusiones

Como se pudo observar a lo largo del experimento, en este caso el modelo de regresión logística es el que mejores resultados generales arrojó. Obtuvo la mejor calificación de F1-Score y de Recall, siendo su punto débil al precisión, lo cual indica que puede estar más propenso a arrojar falsos positivos. En las matrices de confusión del conjunto de prueba, se puede observar como el modelo de regresión logística es el que tiene mayores resultados en su diagonal principal.

Es importante recalcar que, a pesar de que se declara Regresión Logística como el mejor modelo, SVM y Random Forest también son modelos bastante competivos, siendo SVM una buena alternativa si se prioriza la precisión y se sacrifica el recall, mientras que Random Forest pone en la mesa un equilibrio razonable entre precisión y recall, sin ser específicamente el mejor en ambos.

Algo que parece curioso es la gran cantidad de falsos positivos que todos los modelos encuentran en la clase , siendo la columna con mayores valores en la gran mayoría de los casos.