Universidad de La Habana

Facultad de Matemática y Computación

Predicción de Problemas de Codeforces

Juan Carlos Espinosa Delgado C-411
Raudel Alejandro Gómez Molina C-411
Alex Sierra Alcalá C-411
Yoan René Ramos Corrales C-412

Proyecto en github

1. Introducción

1.1. Motivación

La plataforma Codeforces es una herramienta fundamental en la comunidad de programación competitiva, diseñada para desarrollar y entrenar habilidades de resolución de problemas. Los problemas en Codeforces no solo desafían a los competidores, sino que también proporcionan una base sólida para aprender y aplicar diversos algoritmos y estructuras de datos. Cada problema está asociado a una serie de categorías o etiquetas que ayudan a identificar los tipos de algoritmos y técnicas necesarias para resolverlos. Pero este proceso suele estar muy apegado a la a la solución final del ejercicio por lo que la correcta identificación de los tags suele ser un gran reto para los competidores.

Es válido aclarar que la automatización de este proceso no es interés de la programación competitiva, ya que en estos escenarios lo que se busca es el razonamiento lógico de los competidores, en este contexto si pudiera ser interesante el desarrollo de un mecanismo de generación de problemas, pero este no es el objetivo de este proyecto.

Sin embargo la experimentación con la detección de tags en escenarios controlados como este, manteniéndonos en el entorno de que esta tarea es útil para la posterior solución del problema pudiera servir de base para proponer mecanismo de detección de tags en problemas reales orientados al campo de la generación de algoritmos.

1.2. Problemática

Cada problema en la plataforma Codeforces y un problema de programación competitiva en general cuenta primeramente con un título y una descripción en la cual se plantea el objetivo y la problemática del mismo, este segundo componente del problema es la principal fuente de interés para la detección de tags ya que estos se encuentran explícitos en forma de lenguaje natural en dicha descripción.

Pero además de la descripción los problemas cuentan con una restricción de tiempo y espacio de memoria donde se deben desenvolver los algoritmos que den solución al problema (un algoritmo correcto para todos los casos de prueba que no este dentro de los límites establecidos no es considerado como solución). Ahora bien esta restricción también influye en los tags del algoritmo ya el conjunto de tags del problema que satisfacen estas restricciones y la descripción del problema será subconjunto del conjunto de tags solo asociados a la descripción.

Otra característica interesante que nos pudiera aportar información sobre la naturaleza del problema es analizar el código de una solución aceptada del problema, ya que es posible identificar los tags asociados a dicho código y por tanto un subconjunto de los tags del problema. Es importante analizar que con este enfoque solo podemos obtener un subconjunto de los tags ya que un mismo problema puede tener multiples soluciones y cada solución puede tener asociada distintos tipos de tags.

Por tanto como tenemos de por medio un problema asociado a lenguaje natural y actualmente no hay ningún método asociado medianamente eficaz asociado a este campo que no lleve Machine Learning, la propuesta de la solución descrita en este trabajo empleará algoritmos de Machine Learning los cuales iremos introduciendo a lo largo de este trabajo.

1.3. Objetivos Generales y Específicos

1.3.1. Objetivos Generales

El objetivo general de este proyecto es desarrollar un sistema automatizado utilizando técnicas de Machine Learning para detectar y asignar etiquetas (tags) a los problemas de programación en Codeforces.

1.3.2. Objetivos Específicos

- **Recolección y Preprocesamiento de Datos:**
  - Recolectar un conjunto representativo de problemas de Codeforces, incluyendo sus descripciones, restricciones de tiempo y memoria, y etiquetas existentes.

- Realizar el preprocesamiento del texto de las descripciones para normalizar y limpiar los datos, facilitando su análisis.

- **Desarrollo del Modelo de Machine Learning:**
  - Investigar y seleccionar algoritmos de Machine Learning apropiados para la tarea de clasificación de texto, como KNN, Naive Bayes, y redes neuronales.
  - Entrenar varios modelos utilizando el conjunto de datos preprocesado, ajustando hiperparámetros para optimizar el rendimiento.
- **Evaluación del Modelo:**
  - Evaluar los modelos entrenados utilizando métricas de rendimiento como precisión, recall, F1-score y exactitud.
  - Comparar los resultados de diferentes modelos para identificar el más eficaz en la detección de etiquetas.
  - Comparar los resultados con un modelo de lenguaje (en este caso se usará Chat-GPT 3.5).
- **Implementación y Prueba:**
  - Implementar el modelo seleccionado en un entorno de prueba para evaluar su rendimiento en condiciones reales.
  - Realizar pruebas adicionales para validar la consistencia y robustez del sistema en la asignación de etiquetas.
- **Documentación y Propuesta de Mejora:**
  - Documentar el proceso completo de desarrollo, incluyendo la recolección de datos, preprocesamiento, desarrollo del modelo, evaluación e implementación.
  - Proponer mejoras y futuras líneas de investigación basadas en los resultados obtenidos y las limitaciones encontradas durante el desarrollo del proyecto.

1.3.3. Hipótesis

- **Hipótesis Principal:** Un modelo de Machine Learning bien entrenado puede detectar y asignar etiquetas (tags) a los problemas de programación de Codeforces con una precisión comparable a la de un humano experto.
- **Hipótesis Secundarias:**
  - Los algoritmos de clasificación de texto basados en redes neuronales (como LSTM o Transformers) ofrecerán un mejor rendimiento en la detección de etiquetas en comparación con algoritmos más tradicionales como Naive Bayes o KNN.
  - El análisis y utilización de las restricciones de tiempo y memoria, así como del código de soluciones aceptadas, pueden mejorar significativamente la precisión del modelo en la asignación de etiquetas.

1.3.4. Preguntas Científicas

- ¿Qué algoritmos de Machine Learning son más efectivos para la tarea de detección de etiquetas en problemas de programación competitiva?
- ¿Cómo influye la calidad y cantidad de los datos de entrenamiento en el rendimiento del modelo?
- ¿En qué medida las restricciones de tiempo y memoria impactan en la precisión de la detección de etiquetas?
- ¿Es posible mejorar la detección de etiquetas utilizando información adicional del código de soluciones aceptadas?
- ¿Cuáles son las principales limitaciones y desafíos al aplicar Machine Learning en la detección de etiquetas en problemas de programación competitiva?

2. Análisis de los datos

Como dataset usaremos un conjunto de problemas de Codeforces con su identificador, descripción, conjunto de tags, puntos y rating.

2.1. Exploración de datos

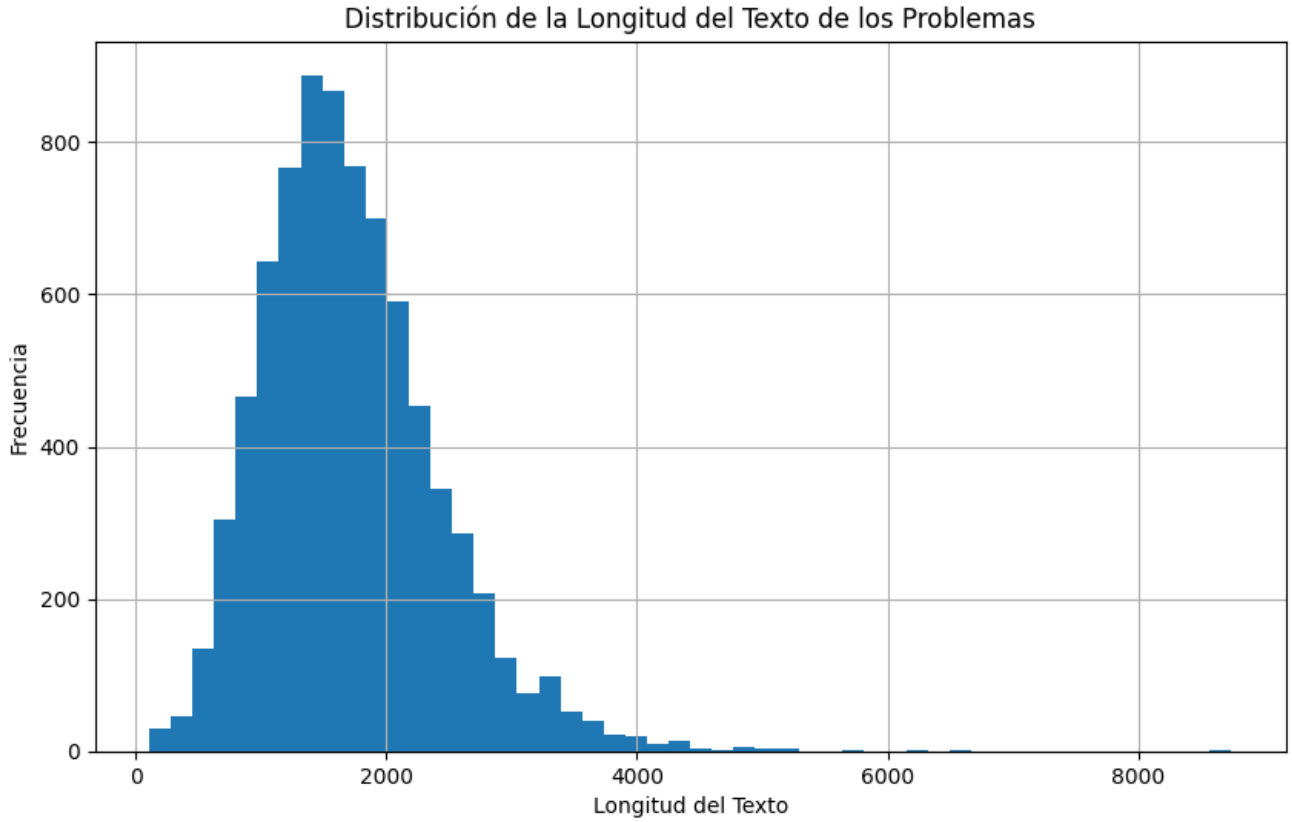
Para esto primero exploramos los datos buscando filas con ausencia de datos y eliminamos datos que no son de interés para nuestro problema.

En este caso hemos identificado que las columnas asociadas a los puntos y el rating cuentan con valores neuronales y no representan interés para nuestro problema por lo que hemos decidido eliminarlas.

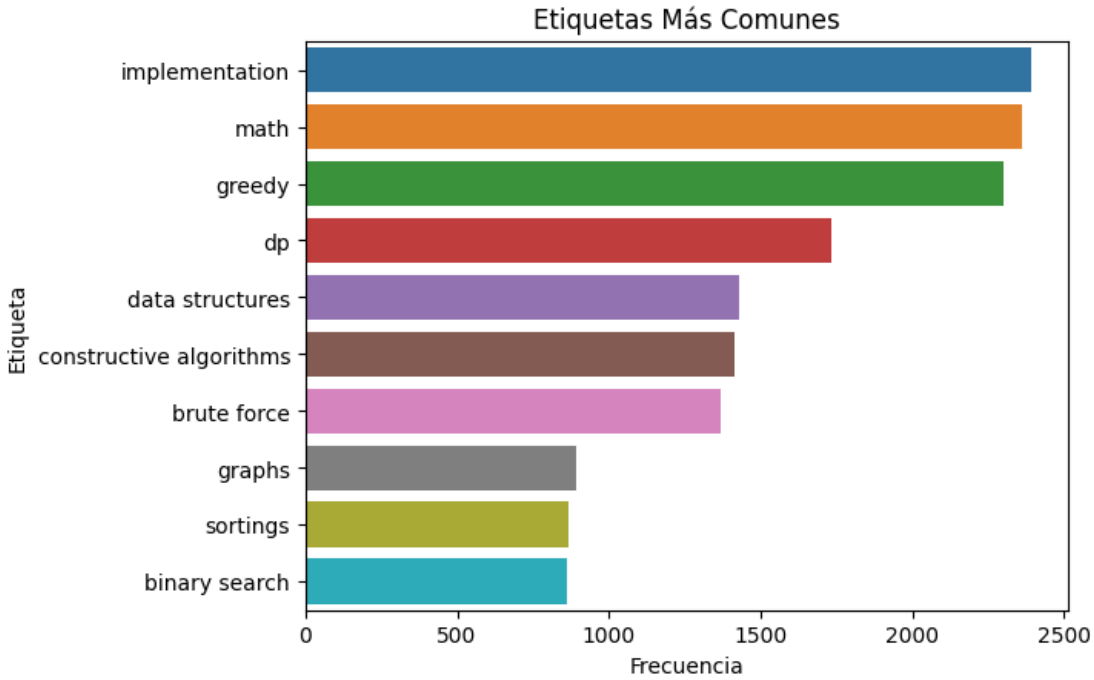
Luego analizamos el contenido de la descripción de los problemas y identificamos el idioma de las mismas ya que esta información puede ser útil para el futuro análisis empleando modelos de lenguaje.

2.1.1. Información recopilada sobre los datos

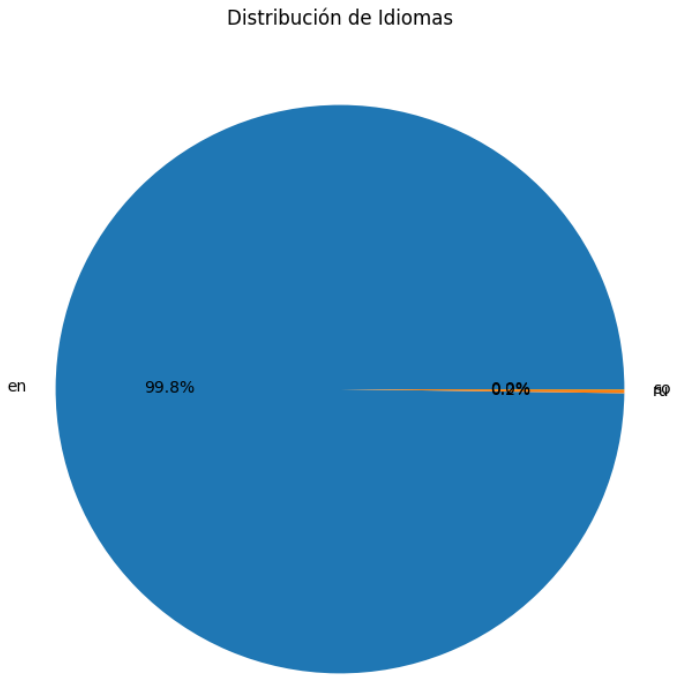
- Distribución de la longitud del texto de los Problemas



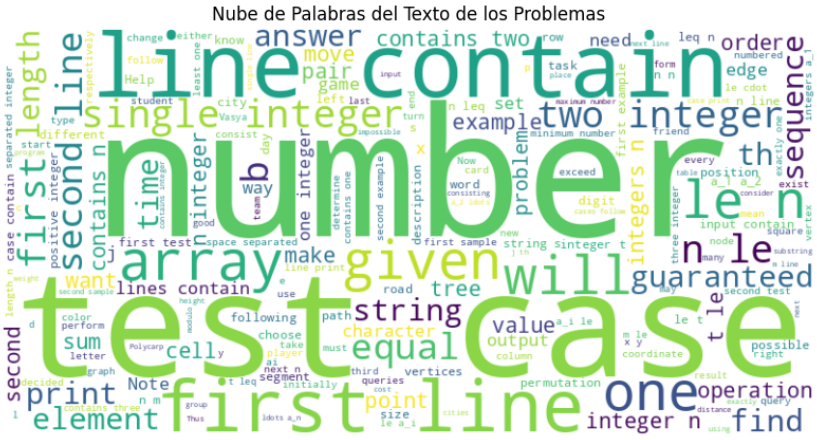
- Frecuencia de las etiquetas más comunes



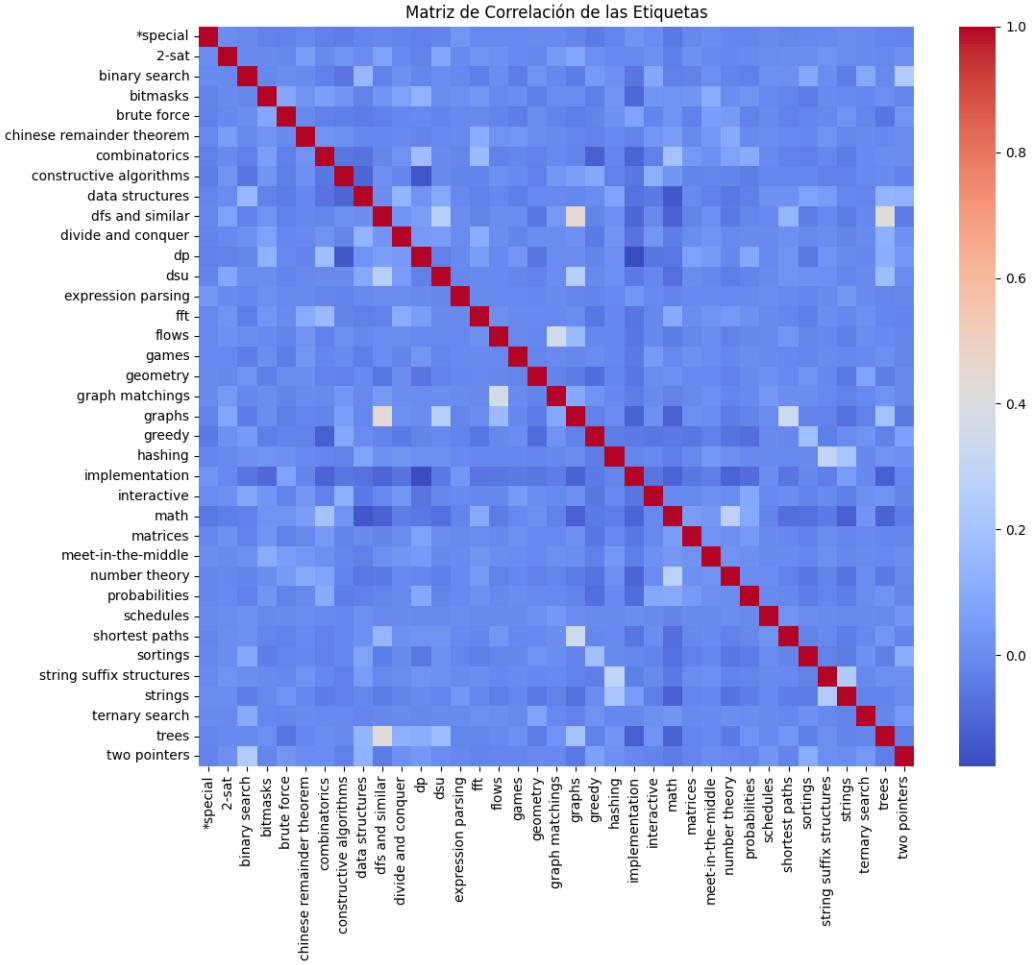
- Distribución de idiomas



- Nube de palabras asociada a la descripción de los problemas



- Matriz de correlación de las etiquetas



2.2. Preprocesamiento de datos

3. Estado del arte / Preliminares

Debe incluir una revisión de la literatura sobre el problema y problemas similares.

4. Propuestas de solución

- Clasificación Multietiqueta con TF-IDF y Naive Bayes

5. Experimentación y resultados

5.1. Clasificación Multietiqueta con TF-IDF y Naive Bayes

5.1.1. Introducción

La clasificación multietiqueta es una variante de la clasificación en la que cada instancia puede pertenecer a múltiples clases simultáneamente. En este estudio, utilizamos la vectorización TF-IDF para la extracción de características y un clasificador Naive Bayes para la clasificación. El objetivo principal es evaluar el rendimiento del modelo en varias métricas de evaluación.

5.1.2. Metodología

- Preprocesamiento de Datos

El conjunto de datos preprocesado pasa a utilizar la vectorización TF-IDF para convertir los datos de texto en características numéricas. Las etiquetas se binarizaron utilizando `MultiLabelBinarizer` para adaptarse a la naturaleza multietiqueta del problema.

- Entrenamiento del Modelo

La clasificación se realizó utilizando un clasificador Naive Bayes dentro de un marco One-vs-Rest. El conjunto de datos se dividió en conjuntos de entrenamiento y prueba utilizando una división 80-20. Una vez el modelo predecía habían problemas a los cuales no se les asignaba ninguna etiqueta, cosa que no sucede en el codeforces, por lo cual se le hace asignar a dichos problemas la eqtiqueta más probable, garantizando así que todo problema contenga al menos una etiqueta.

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(text_data)
mlb = MultiLabelBinarizer()
y = mlb.fit_transform(labels)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

clf = OneVsRestClassifier(MultinomialNB())
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

5.1.3. Resultados

- Informe de Clasificación

```
classification_report(y_test, y_pred, target_names=mlb.classes_)
```

```
                      precision    recall  f1-score   support

          bruteforce       0.20      0.02      0.03        61
constructivealgorithms       0.50      0.01      0.03        73
        datastructures       1.00      0.04      0.08        49
         dfsandsimilar       0.00      0.00      0.00         4
                    dp       0.00      0.00      0.00        50
              geometry       1.00      0.18      0.30        17
                greedy       0.52      0.41      0.46       111
        implementation       0.52      0.69      0.59       142
                  math       0.63      0.29      0.39       101
               strings       0.83      0.37      0.51        27

             micro avg       0.55      0.30      0.39       635
             macro avg       0.52      0.20      0.24       635
          weighted avg       0.52      0.30      0.32       635
           samples avg       0.54      0.33      0.39       635
```

- Accuracy

```
Overall Accuracy: 14.46\%
```

- Matriz de Confusión

Se tiene un jupyter interactivo en el que se puede consultar la matriz de confusión para cada etiqueta

- Precision ,F1-Score ,Recall y Support para cada etiqueta

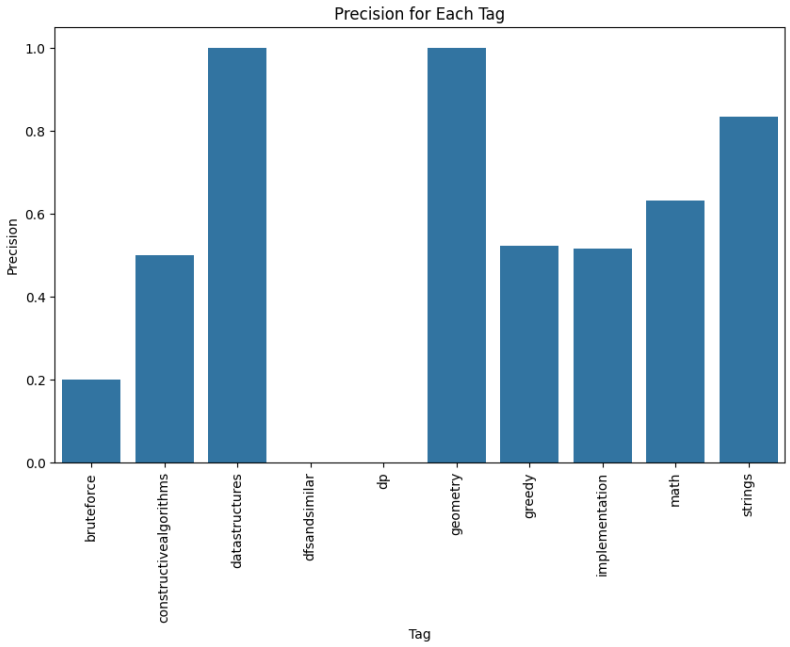


Figura 1: Precision

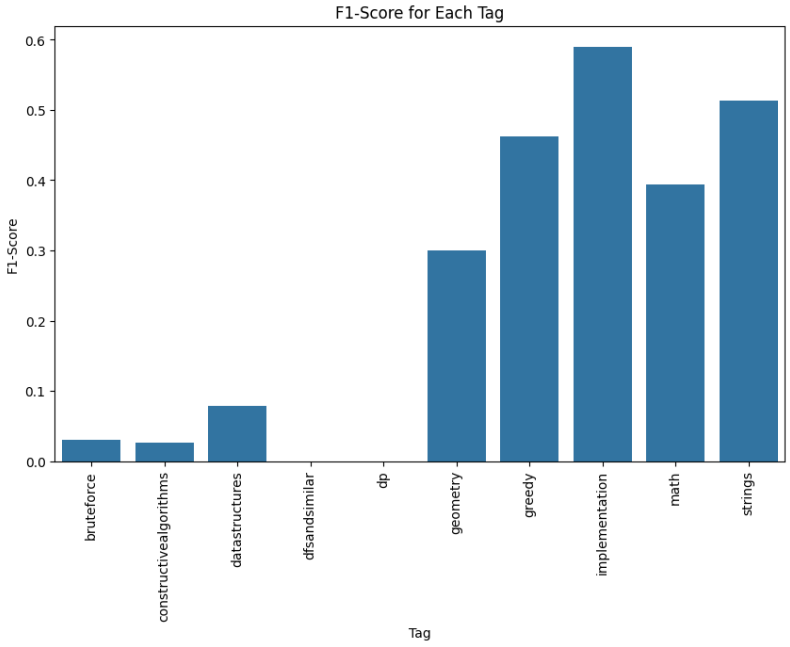


Figura 2: F1-Score

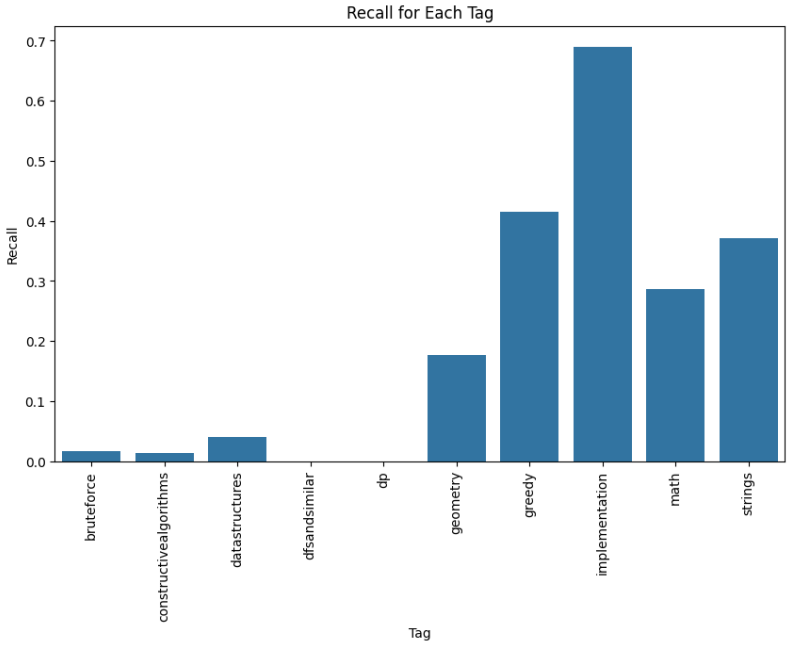


Figura 3: Recall

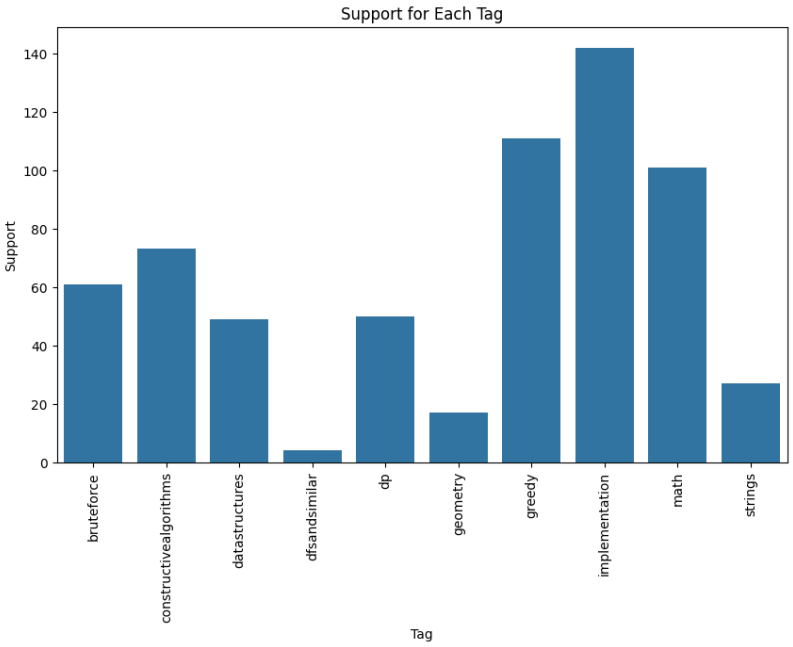


Figura 4: Support

5.2. Clasificación Multietiqueta con TF-IDF y KNN

5.2.1. Introducción

Al igual que en la sección anterior, usaremos el enfoque de modelo de aprendizaje multietiquetas. La vectorización usada es la misma que en Naive-Bayes, TF-IDF para extraer características y un clasificador KNN para la clasificación. El objetivo poder comparar el rendimiento de este modelo con los otros que hemos usado.

5.2.2. Metodología

- Preprocesamiento de Datos

además de preprocesar la descripción de los problemas para usar TF-IDF, al ser KNN un modelo que solo se puede entrenar con valores numéricos, también se binarizaron los tags de los problemas, y se eliminaron columnas poco relevantes como el idioma y la identificación de los problemas.

- Entrenamiento del Modelo

La clasificación se realizó utilizando un clasificador KNN dentro de un marco MultiOutput. El conjunto de datos se dividió en conjuntos de entrenamiento y prueba utilizando una división 80-20.
Pero cuál sería el mejor valor de K para el modelo, para esto se realizó una búsqueda de hiperparámetros con valores de K de 1 a 100, dando como resultado que K = 12 era el valor con mayor precisión en los resultados.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

neighbors = np.arange(1, 100)
train_accuracies = {}
test_accuracies = {}
reports = []
for neighbor in neighbors:
    knn = KNeighborsClassifier(n_neighbors=neighbor)
    mlb_knn = MultiOutputClassifier(knn)
    mlb_knn.fit(X_train, y_train)
    ypred = mlb_knn.predict(X_test)
    train_accuracies[neighbor] = mlb_knn.score(X_train, y_train)
    test_accuracies[neighbor] = mlb_knn.score(X_test, y_test)
    reports.append(classification_report(y_test, ypred, zero_division = 0))
```

5.2.3. Resultados

- Informe de Clasificación

```
report = classification_report(y_test, y_pred, target_names=list(all_tags))
```

	precision	recall	f1-score	support
math	0.410256	0.163265	0.233577	98.000000
strings	0.555556	0.500000	0.526316	30.000000
datastructures	0.900000	0.209302	0.339623	43.000000
greedy	0.500000	0.350000	0.411765	100.000000
dfsandsimilar	0.000000	0.000000	0.000000	10.000000
constructivealgorithms	0.100000	0.017544	0.029851	57.000000
implementation	0.602740	0.325926	0.423077	135.000000
geometry	1.000000	0.062500	0.117647	16.000000
bruteforce	0.500000	0.015152	0.029412	66.000000
dp	0.000000	0.000000	0.000000	47.000000

- Accuracy

```
Overall Accuracy: 8.00\%
```

- Precision ,F1-Score ,Recall y Support para cada etiqueta

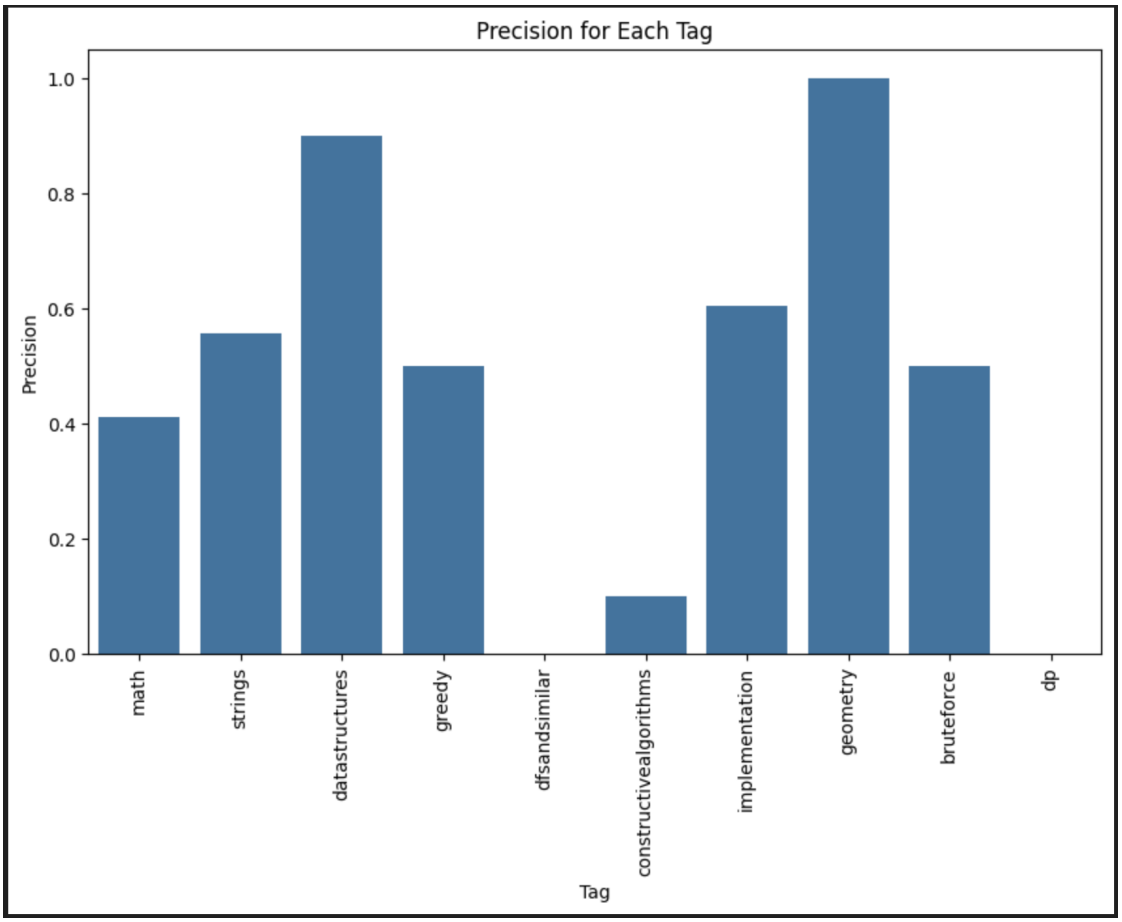


Figura 5: Precision

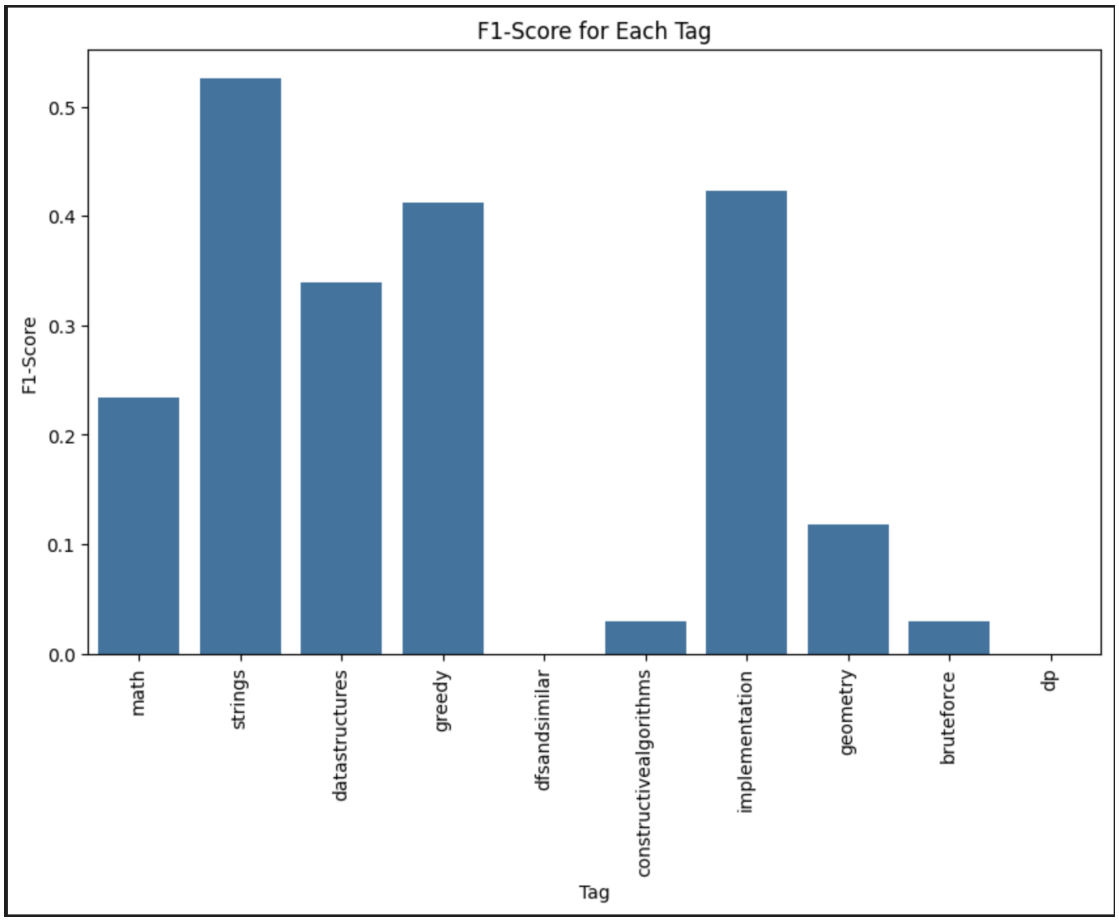


Figura 6: F1-Score

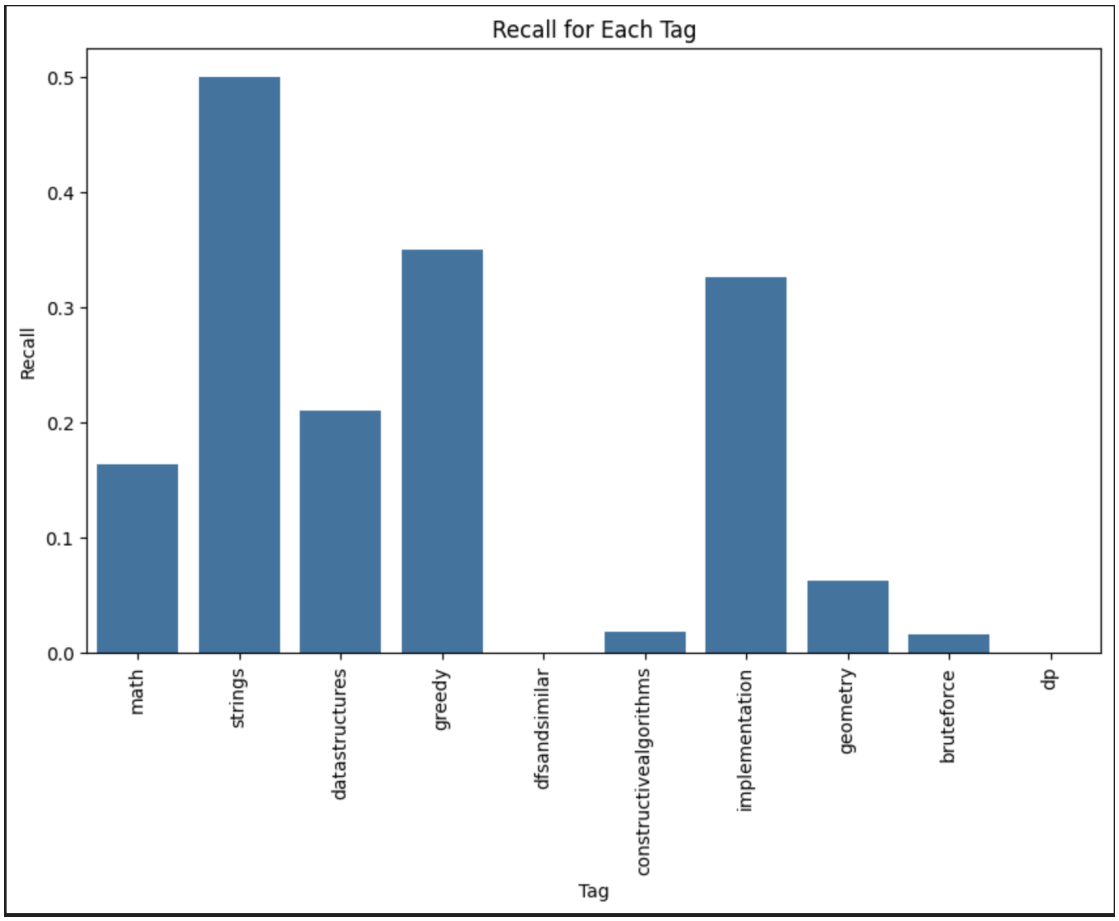


Figura 7: Recall

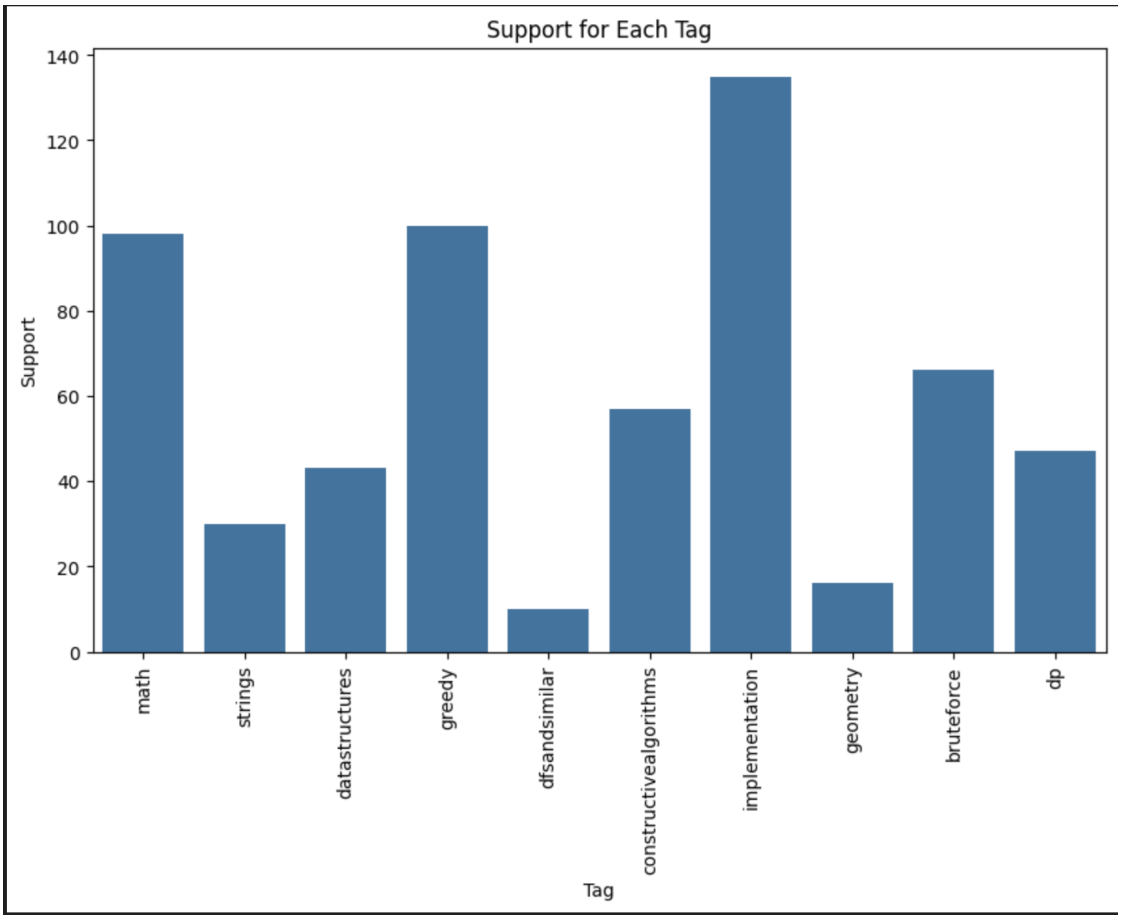


Figura 8: Support

5.3. Chat GPT

Para analizar que tan bien se comportaba Chat GPT 3.5, prediciendo los tags de Codeforces, usamos prompt engineering con un preprocesamiento sobre los problemas:

```
def prompt(description):
    all_tags_str = ', '.join(all_tags)
    return f'Give this set of {all_tags_str} tags and this problem ${description}, give me the set
    of problem tags in the following format: greedy, implementation, dp'
```

Listing 1: Prompt para tagear los problemas

	Metric	ChatGPT	Naive Bayes	KNN
0	Accuracy	0.020000	0.146000	0.080000
1	F1 (macro)	0.002431	0.240000	0.210000
2	F1 (micro)	0.020000	0.390000	0.290000
3	F1 (weighted)	0.017556	0.320000	0.260000
4	F1 (samples)	0.002431	0.390000	0.240000

6. Discusión de los resultados

Debe incluir la repercusión ética de las soluciones.

7. Conclusiones y trabajo futuro