Minería de Medios Sociales - 2024-2025

Universidad de Granada

Práctica de Análisis de Sentimientos - KNIME

Miguel García López

Índice

Índice de figuras

Índice de cuadros

1. Introducción

Este documento aborda el tema del **análisis de sentimientos** y describe algunas de las técnicas utilizadas a lo largo de todo el proceso. El flujo de trabajo, o *workflow*, se desarrollará utilizando el *software KNIME*, que integra una amplia variedad de herramientas específicas para este propósito, además de permitir la incorporación de *plugins* adicionales si se requiere. Una de las principales ventajas de *KNIME* es su enfoque visual para la construcción de procesos *ETL*, lo que facilita la comprensión y el seguimiento del flujo de interacción con los datos.

Para este trabajo se requiere crear varios flujos usando *Lexicons* concretos para calcular la orientación de sentimientos de una serie de *reviews* de **IMDb**. Para ello se utilizaran los *Lexicons*:

1. **SentiWordNet 3.0**.
2. **SenticNet 5**
3. **Subjectivity Clues (Subjclue)**

Además se deberá comparar los resultados obtenidos con las predicciones resultantes del uso de *Lexicons*, con los resultados obtenidos de clasificadores de aprendizaje automático, como por ejemplo **SVM**, **Logistic Regression** u otros.

2. Conjunto de datos

Se utiliza como conjunto principal de datos el *dataset* de **IMDb** (figura 1), el cual contiene un conjunto de reseñas etiquetadas con sentimientos negativos, positivos y neutros, así como el texto correspondiente a la reseña y su *URL*. Para el trabajo concreto que se lleva a cabo en este documento, se filtran aquellas reseñas con clasificación neutra, ya que se van a evaluar solo clasificaciones binarias con el uso de *Lexicons*.

RowID	Index *Number (integer)*	URL *String*	Text *String*	Sentiment *String*
Row0	3617	http://www.imdb.com/title/tt0210075/usercomments	Girlfight follows a project dwelling New York high school girl from a ser	POS
Row1	3671	http://www.imdb.com/title/tt0337640/usercomments	Hollywood North is an euphemism from the movie industry as they wer	POS
Row2	3157	http://www.imdb.com/title/tt0303549/usercomments	That '70s Show is definitely the funniest show currently on TV. I started	POS
Row3	660	http://www.imdb.com/title/tt0716825/usercomments	9/10- 30 minutes of pure holiday terror. Okay, so it's not that scary. But	POS
Row4	265	http://www.imdb.com/title/tt0182225/usercomments	A series of random, seemingly insignificant thefts at her sister's boardi	POS
Row5	4027	http://www.imdb.com/title/tt0347779/usercomments	A very good adaptation of the novel by amrita pritam. Urmila and mano	POS
Row6	5820	http://www.imdb.com/title/tt0095655/usercomments	Ah, Moonwalker, I'm a huge Michael Jackson fan, I grew up with his mu	POS
Row7	1574	http://www.imdb.com/title/tt0298131/usercomments	Although the beginning of the movie in New York takes too long, the m	POS
Row8	10668	http://www.imdb.com/title/tt0088915/usercomments	As many reviewers here have noted, the film version differs quite a bit f	POS
Row9	1473	http://www.imdb.com/title/tt0828154/usercomments	Bear in mind, any film (let alone documentary) which asserts any kind c	POS
Row10	8337	http://www.imdb.com/title/tt0110099/usercomments	Being a big fan of the romantic comedy genre, and therefore having see	POS
Row11	11217	http://www.imdb.com/title/tt0098492/usercomments	Being an otaku since the days of Robotech, I can still say that Gunbuste	POS

Figura 1: Muestra del conjunto de datos de reseñas de **IMDb**

3. Lexicons

Puede observarse el flujo completo realizado por el estudiante en la figura 2. El primer nodo `csv_reader` es el que lee el conjunto de datos. Posterior a su lectura se realiza un pre-procesamiento básico en el que se eliminan los signos de puntución, se pasa de tipo de dato *string* a *document* y se pasa todo el texto a minúscula. Con ello ya se tiene el texto listo para pasar al proceso de *tagging*.

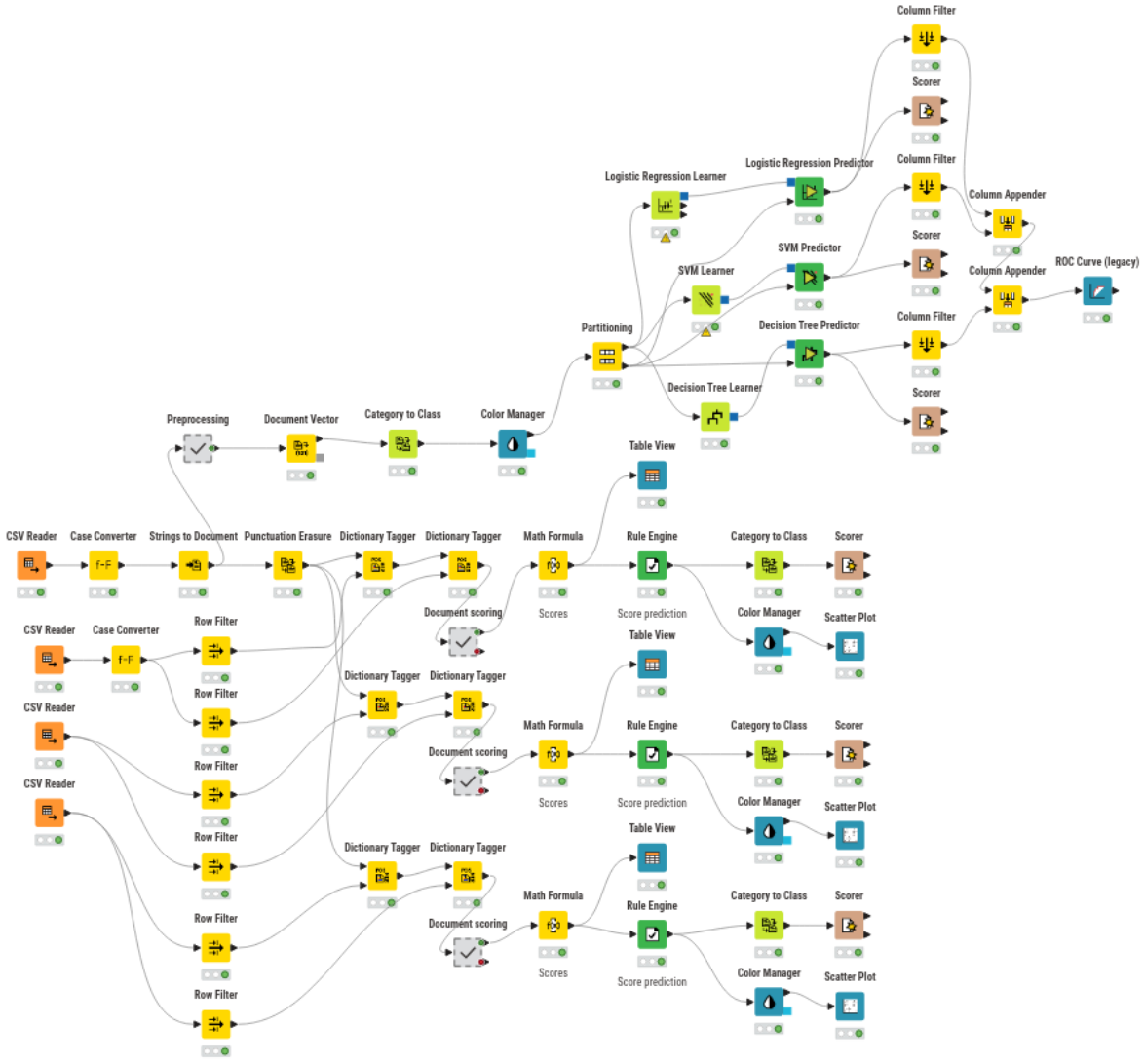

Figura 2: Flujo completo de *KNIME*

Los nodos lectores de archivos *csv* debajo del lector del conjunto de datos, se encargan de cargar los *Lexicons*, previamente procesados por un *script* de *Python* que ya los pasa a minúscula, limpia duplicados, manipula alguna palabra mal formada quitando signos como "_", etc. Se proporciona el código a continuación:

```
    def clean_word(word):
        # Replace underscores with spaces
```

```
        word = word.replace("_", " ")
        # Remove accents
        word = unicodedata.normalize("NFD", word)
        word = word.encode("ascii", "ignore").decode("utf-8")
        # Convert to lowercase
        word = word.lower()
        return word

    def process_sentiwordnet_file(input_path, output_path):
        seen_words = set()
        with open(output_path, "w", newline="", encoding="utf-8") as csvfile:
            csv_writer = csv.writer(csvfile)
            csv_writer.writerow(["word", "polarity", "score"])

            with open(input_path, "r", encoding="utf-8") as infile:
                for line in infile:
                    if line.startswith("#") or line.strip() == "":
                        continue

                    parts = line.strip().split("\t")
                    if len(parts) >= 6:
                        pos, id_num, pos_score, neg_score, synset_terms, *gloss_parts = parts
                        pos_score = float(pos_score)
                        neg_score = float(neg_score)

                        if pos_score > 0 or neg_score > 0:
                            words = synset_terms.split()
                            for word in words:
                                word_clean = word.split("#")[0]
                                word_clean = clean_word(word_clean)

                                if word_clean not in seen_words:
                                    seen_words.add(word_clean)
                                    if pos_score >= neg_score:
                                        csv_writer.writerow([word_clean, "positive", pos_score])
                                    else:
                                        csv_writer.writerow([word_clean, "negative", neg_score])

    def process_subjclue_file(input_path, output_path):
        seen_words = set()
        with open(output_path, "w", newline="", encoding="utf-8") as csvfile:
            csv_writer = csv.writer(csvfile)
            csv_writer.writerow(["word", "polarity"])

            with open(input_path, "r", encoding="utf-8") as infile:
                for line in infile:
                    line = line.strip()
                    if not line:
                        continue
```

```
52
53                      # Extract word and polarity from line
54                      word_match = re.search(r"word1=(\w+)", line)
55                      polarity_match = re.search(r"priorpolarity=(\w+)",
    line)
56
57                      if word_match and polarity_match:
58                          word = word_match.group(1)
59                          polarity = polarity_match.group(1)
60
61                          word_clean = clean_word(word)
62
63                          if word_clean not in seen_words:
64                              seen_words.add(word_clean)
65                              csv_writer.writerow([word_clean, polarity])
66
67      def process_senticnet_file(input_path, output_path):
68          seen_words = set()
69          with open(output_path, "w", newline="", encoding="utf-8") as
    csvfile:
70              csv_writer = csv.writer(csvfile)
71              csv_writer.writerow(["word", "polarity", "score"])
72
73              with open(input_path, "r", encoding="utf-8") as infile:
74                  # Skip the header line
75                  header = next(infile, None)
76
77                  for line in infile:
78                      line = line.strip()
79                      if not line:
80                          continue
81
82                      # Use regex to split the line into parts, handling
    variable whitespace
83                      parts = re.split(r"\s{2,}|\t+", line)
84
85                      if len(parts) >= 3:
86                          concept = parts[0].strip()
87                          polarity = parts[1].strip().lower()
88                          intensity = parts[2].strip()
89
90                          # Clean the concept word
91                          word_clean = clean_word(concept)
92
93                          # Convert intensity to float
94                          try:
95                              intensity_float = float(intensity)
96
97                              if word_clean not in seen_words:
98                                  seen_words.add(word_clean)
99                                  csv_writer.writerow([word_clean,
    polarity, intensity_float])
100                         except ValueError:
101                             print(f"Skipping invalid intensity value: {
```

```
    intensity} for word: {word_clean}")
102
```

Todos los *Lexicons* son filtrados y procesados por sentimiento negativo y positivo, para posteriormente utilizar el nodo `dictionary_tagger` para relacionaer *tags* de sentimiento con cada texto por ocurrencia de palabras de los *Lexicons*. Después pasan por un meta-nodo (figura 3) de procesado que se encarga de extraer un *score* que representa la clasificación final del texto mediante una división de ocurrencias positivas y negativas. Posterior a todo ese flujo, se clasifica utilizando dos simples reglas: Si el *score* es positivo, el texto es positivo, si es negativo el texto es negativo.

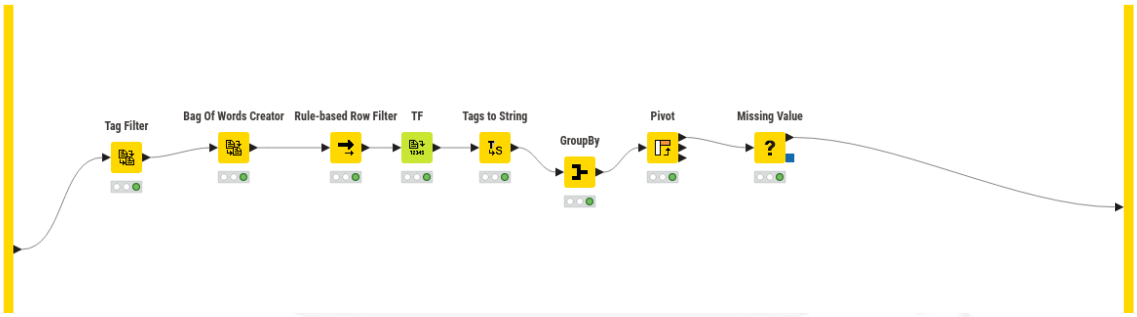

Figura 3: Meta-nodo de procesamiento de los *Lexicons* + *dataset* para el *tagging*

3.1. Resultados

Los resultados obtenidos por cada *Lexicon* (figuras 4, 5, 6) son, en términos generales, decentes. Concretamente *senticnet* arroja unos resultados totalmente desbalanceados, siendo capaz de identificar casi siempre los comentarios positivos, pero nunca los negativos. Esto es resultado de, seguramente, un desbalance enorme en el conjunto de términos, que lleva a predecir prácticamente siempre la clase positiva. Los otros son son mucho más equilibrados, pero siguen fallando por centenares, por lo que no son demasiado útiles.

Se observa gráficamente en las figuras de puntos 10 las clasificaciones correspondientes a cada diccionario de términos para el conjunto de datos de reseñas.

4. Clasificadores ML

Paralelo al procesamiento de los *tags* se encuentra el flujo de los clasificadores de aprendizaje automático, concretamente los modelos **Support Vector Machine**, **Logistic Regression** y **Decision Tree**.

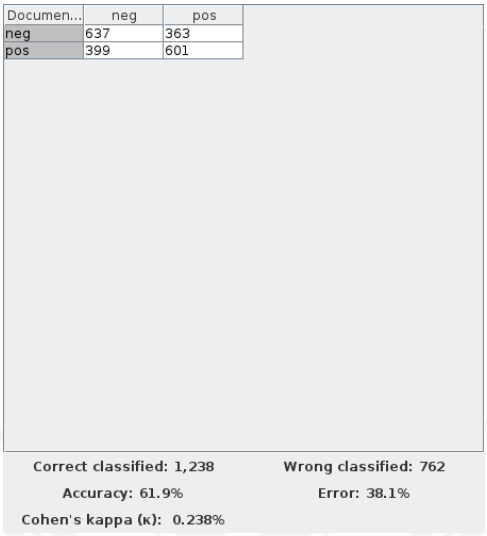

Figura 4: Matriz de confusión para *Lexicon* de *sentiwordnet*

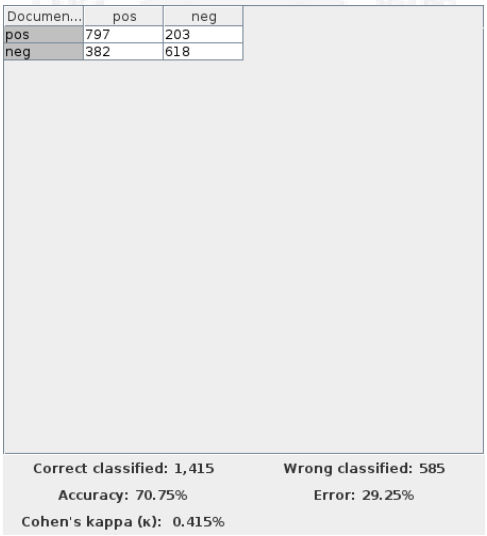

Figura 5: Matriz de confusión para *Lexicon* de *subjclue*

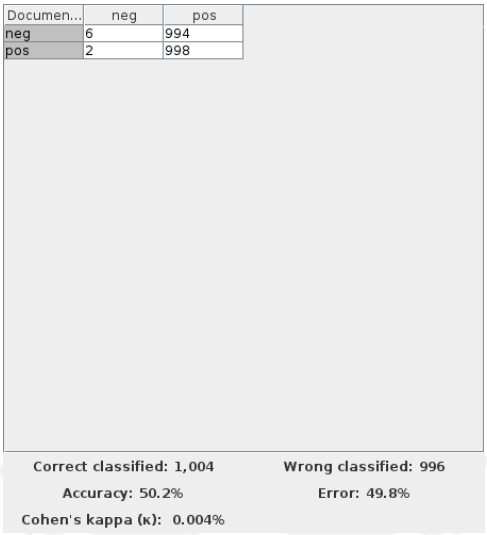

Figura 6: Matriz de confusión para *Lexicon* de *senticnet*

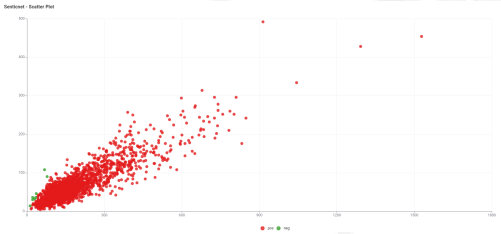

Figura 7: Senticnet - Scatter Plot

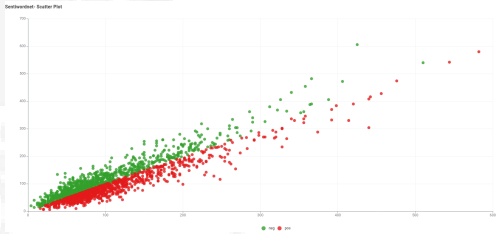

Figura 8: Sentiwordnet - Scatter Plot

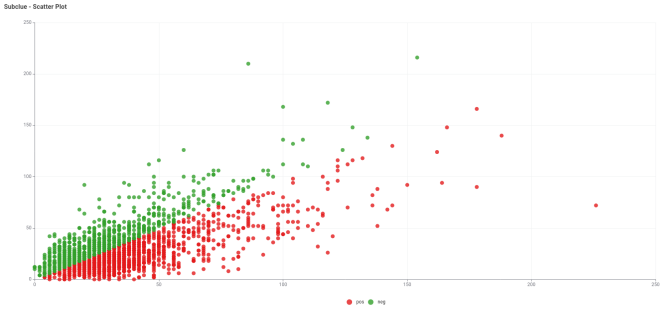

Figura 9: Subclue - Scatter Plot

Figura 10: Clasificación de cada *Lexicon*

Para el uso de estos algoritmos, es necesario vectorizar los datos de forma que puedan ser usados como entrada. Se usa por este motivo el nodo `document_vector`, el cual convierte documentos en vectores numéricos, basándose simplemente en si una palabra aparece o no (binario). Cada dimensión del vector corresponde a una palabra (o n-grama) del vocabulario.

A continuación se hace una breve descripción de cada algoritmo utilizado. Se ha probado a cambiar los hiper-parámetros de los algoritmos, pero finalmente el estudiante decidió dejar los valores por defecto.

4.1. Support Vector Machine

El modelo **Support Vector Machine** (SVM) busca encontrar el hiperplano que mejor separa las clases de sentimientos (por ejemplo, positivo y negativo) maximizando el margen entre ellas. En análisis de sentimientos, **SVM** es eficaz debido a su capacidad para manejar espacios de alta dimensión, como los vectores de palabras. Se ha utilizado la versión lineal del algoritmo, que es la que mejores resultados y además es la versión por defecto.

4.2. Logistic Regression

La **Regresión Logística** es un clasificador probabilístico que modela la probabilidad de pertenencia a una clase de sentimiento. Es ampliamente utilizado por su simplicidad, interpretabilidad y buen rendimiento en conjuntos de datos linealmente separables.

4.3. Decision Tree

El **Árbol de Decisión** clasifica sentimientos siguiendo una estructura jerárquica de decisiones basadas en las características del texto. Su principal ventaja en análisis de sentimientos es la facilidad para interpretar las reglas que llevan a predecir una emoción o polaridad específica.

4.4. Resultados

En las figuras 11, 12 y 13 se pueden observar los resultados de cada algoritmo. Los tres rinden mucho mejor que la clasificación por *tags* de los *Lexicons*. Las predicciones del trio de clasificadores son muy balanceadas, por lo que son capaces de distinguir entre reseñas positivas y negativas sin problema alguno.

La *Cohen's Kappa* es una métrica que mide el acuerdo entre dos clasificadores (o dos evaluadores) que asignan etiquetas a un conjunto de datos. Tanto **SVM** como **LR** obtiene valores para esta métrica en torno a 0,7, lo que significa que predicen bastante bien y el acuerdo entre predicción y realidad es considerablemente mejor de lo que se lograría por azar.

El mejor modelo es el árbol de decisión, cuyo *cohen's kappa* es de 0,85 (es muy consistente) y tiene una precisión del 90 %.

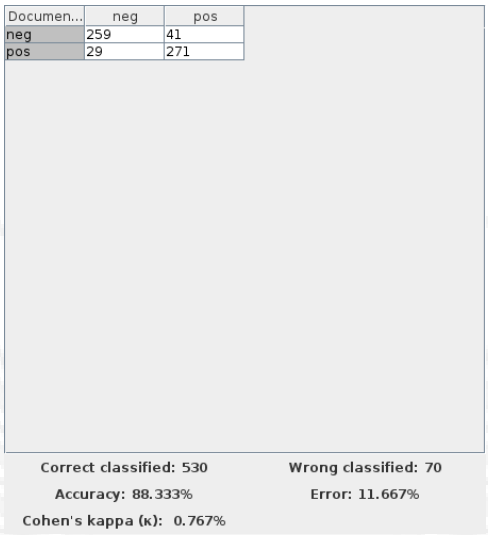

Figura 11: Matriz de confusión para **Logistic Regression**

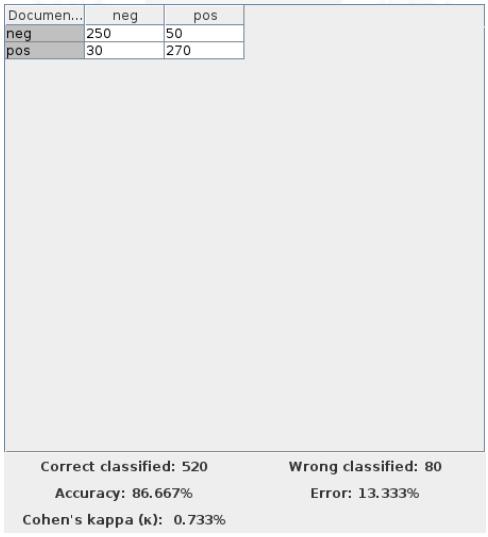

Figura 12: Matriz de confusión para **SVM**

La curva **ROC** se genera trazando la tasa de verdaderos positivos contra la tasa de falsos positivos para diferentes umbrales de decisión. El área bajo esta curva (**AUC-ROC**) es una medida de la capacidad discriminativa del modelo. Al ajustar el umbral de

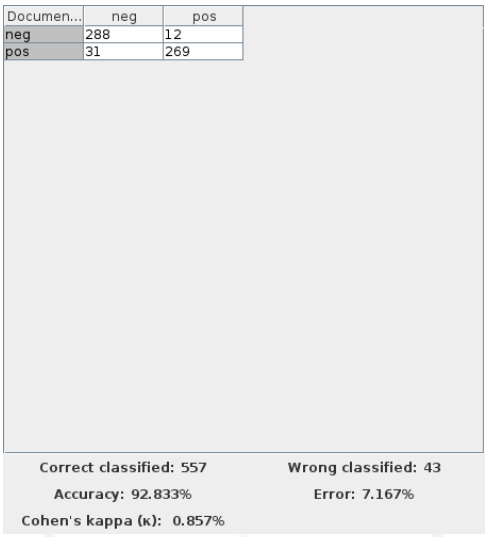

Figura 13: Matriz de confusión para **Decision Tree**

decisión, que por defecto suele ser 0,5 en muchos clasificadores, se puede priorizar según las necesidades de la aplicación:

- Mayor sensibilidad (*recall*): reducir el número de falsos negativos.
- Mayor especificidad: reducir el número de falsos positivos.

En la figura 14 se observan los valores de área bajo la curva **ROC** para los tres clasificadores. Los tres obtienen valores casi idénticos, por lo que se puede concluir, junto a la información previamente analizada, que separan perfectamente las clases positivas y negativas.

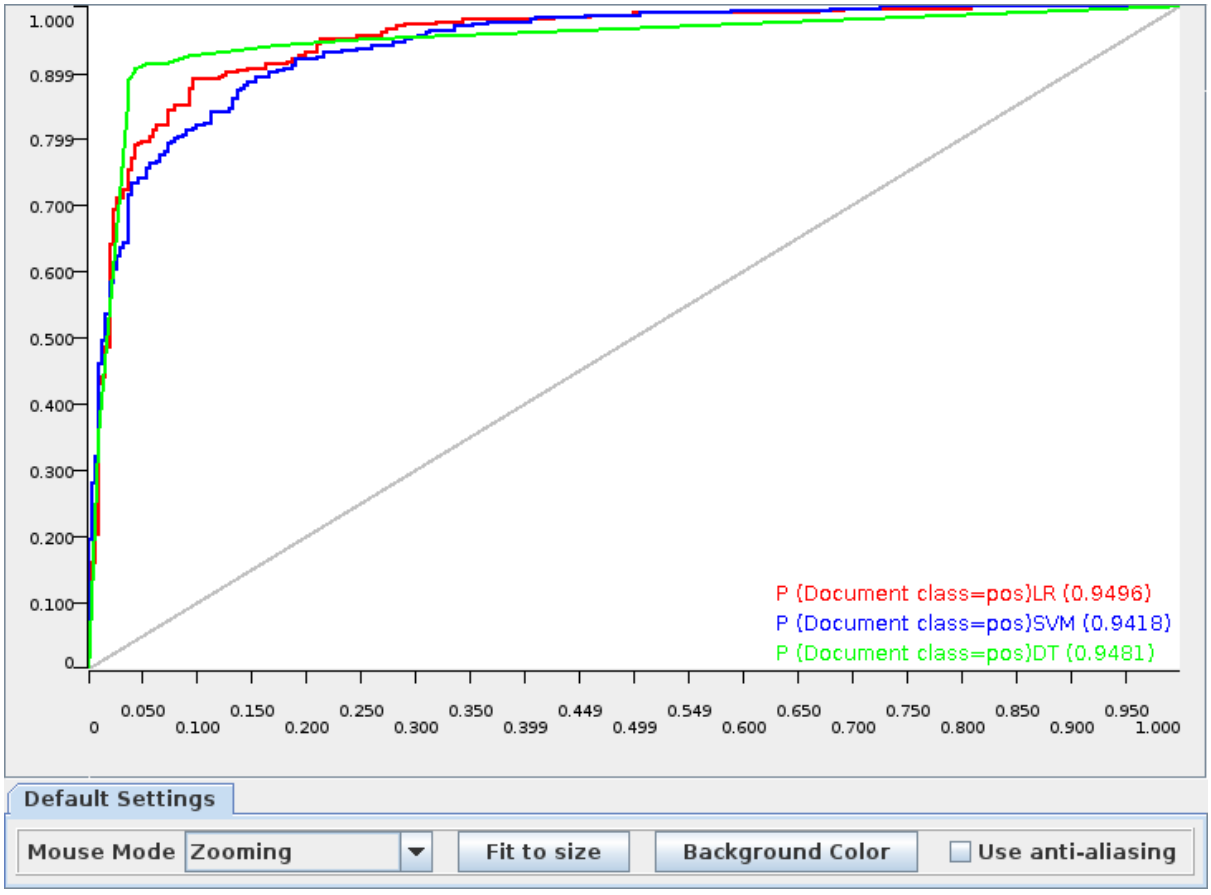

Figura 14: Visualización de curvas *ROC* para los tres clasificadores

5. Conclusión

El análisis de sentimientos mediante diferentes técnicas ha demostrado resultados significativamente diversos. Los enfoques basados en *Lexicons*, aunque conceptualmente simples, presentaron limitaciones importantes. Especialmente *senticnet* mostró un sesgo notable hacia las clasificaciones positivas, mientras que las otras dos versiones ofrecieron resultados más equilibrados pero aún insuficientes para aplicaciones prácticas.

En contraste, los algoritmos de aprendizaje automático demostraron ser claramente superiores. El Árbol de Decisión destacó con una precisión del 90 % y un coeficiente *Cohen's Kappa* de 0,85. Las curvas **ROC** confirmaron la excelente capacidad discriminativa de estos modelos, con áreas bajo la curva prácticamente idénticas.

Este estudio evidencia que, para el análisis de sentimientos en reseñas de **IMDb**, los métodos basados en aprendizaje automático superan significativamente a los enfoques léxicos tradicionales.