TP2: Críticas Cinematográficas - Grupo 4

Introducción

Descripción del problema

El objetivo de este trabajo fue desarrollar modelos de aprendizaje automático con el fin de poder clasificar críticas cinematográficas como positivas o negativas. El trabajo fue enmarcado dentro de una competencia en Kaggle, en donde se evaluaron los rendimientos de cada modelo.

Características del dataset

Tamaño del dataset (train.csv): 50.000 registros
Columnas:

- ID: identificador único
- review_es: texto de la crítica
- sentimiento: etiqueta (positivo - negativo)

Observaciones:

- Mucha variedad de las críticas en cuanto a longitud
- Está balanceado
- No hay valores nulos

Técnicas exploradas y pruebas realizadas

Técnicas exploradas por modelo::

- **Naiva Bayes:** Tuning del parámetro *alpha*
- **Random Forest:** Múltiples configuraciones de hiperparámetros probadas mediante *RandomizedSearchCV*
- **XGBoost:** Múltiples configuraciones de hiperparámetros probadas mediante *RandomizedSearchCV*
- **Red Neuronal:** Optimizada mediante *KerasTuner*, uso de *Dropout*, *BatchNormalization*, *ReduceLROnPlateau*, etc.
- **Ensamble:** Votación mayoritaria

Además, lógicamente se utilizó una división de entrenamiento/validación para estimar el rendimiento de los modelos previamente a subirlos a Kaggle.

Preprocesamiento Aplicado

Las técnicas de preprocesamiento utilizadas fueron:

- Limpieza de texto:
    - Conversión a minúsculas
    - Eliminación de URLs
    - Eliminación de signos de puntuación
    - Eliminación de espacios múltiples
    - Eliminación de StopWords
    - Eliminación de reviews en otro idioma (en un dataset alterno)

- Lematización:

- Bag of Words - TF-IDF: Se utilizaron unigramas y bigramas (*ngram_range=(1, 2)*)

- Embeddings: Se entrenó un modelo propio de Word2Vec utilizando el algoritmo SkipGram(sg=1). Se usaron vectores de tamaño 100(vector_size=100) y se consideraron palabras de contexto aquellas que estén a 3 palabras de distancia(window=3).

- Conversión de etiquetas: Cuando fue necesario, el campo binario fue convertido a valores binarios numéricos.

Hipótesis y supuestos

1. El sentimiento de una crítica puede ser inferido únicamente a partir del texto, sin metadatos adicionales.
2. La inclusión de bigramas en la vectorización mejora la detección de contexto y matices del lenguaje.
3. Las técnicas de regularización ayudan a mejorar la generalización de la red neuronal.
4. Comparar los F1-Score de entrenamiento y validación es una buena técnica para analizar un posible overfitting.

Cuadro de Resultados

Modelo	F1-Test	Presicion Test	Recall Test	Kaggle
Bayes Naive	0.84	0.87	0.82	0.74161

Random Forest	0.82	0.82	0.82	0.71486
XGBoost	0.84	0.84	0.84	0.75130
Red Neuronal	0.85	0.85	0.85	0.76894
Ensamble	-	-	-	0.77146

Descripción de Modelos

Naive Bayes

- Se utilizó *MultinomialNB*, efectivo para tareas de clasificación de texto como lo es en este caso con representaciones como Bag of Words.
- Técnica de vectorización: *TfidVectorizer* con unigramas y bigramas.
- Hiperparámetro ajustado: *alpha*
    - Se utilizó *GridSearchCV* para explorar los valores: [0.1, 0.5, 1.0, 2.0, 5.0]
- Evaluación: Se obtuvieron resultados aceptables como modelo base siendo rápido de entrenar. El modelo que mejor resultados dio en la competencia (0.74161) fue el que tuvo simplemente un limpieza de texto base, sin eliminación de stopwords ni lematización. El modelo entrenado con el texto lematizado y sin stopwords, dio mejores resultados tanto en entrenamiento como en validación pero dio un resultado inferior en la competencia (0.73444).

Random Forest

- Utilizamos *RandomForestClassifier*, realizando una búsqueda de hiperparámetros con *RandomizedSearchCV*.
- Técnica de vectorización: Misma que Naive Bayes
- Hiperparámetros ajustados:
    - *n_estimators*
    - *max_depth*
    - *min_samples_split*, *min_samples_leaf*
    - *max_features*
- Se aplicó regularización controlando el crecimiento de los árboles para evitar overfitting.
- Evaluación: Al igual que naive bayes, el modelo que mejor resultados dio fue con una limpieza de texto simple sin eliminación de stopwords y sin lematización. También se probó con el uso de embeddings, sin embargo

fue el modelo que peores resultados dio tanto en entrenamiento como en la competencia.

XGBoost

- Se utilizó *XGBClassifier* con búsqueda exhaustiva de hiperparámetros mediante *RandomizedSearchCV*.
- **Técnica de vectorización:** TF-IDF y Embeddings.
- **Hiperparámetros ajustados:**
- *n_estimators, max_depth, learning_rate, subsample, colsample_bytree, gamma, reg_alpha, reg_lambda*.
- Se realizaron diversas versiones con más y menos combinaciones.
- **Evaluación:** Para esté modelo el mejor resultado se obtuvo con el uso de embeddings sin eliminación de stopwords con una clara mejoría que vectorización TF IDF (0.75130 vs 0.70265)

Red Neuronal

- Entrenada sobre vectores TF-IDF densificados y Embeddings.
- KerasTuner para búsqueda automática de hiperparámetros.
- **Arquitectura:**
    - Capa de entrada:
    - Densa (128-512 neuronas, activación relu o swish)
    - BatchNormalization
    - Dropout (0.3 - 0.6)
- **Capas ocultas:**
    - 1 a 2 capas ocultas densas (64 - 256 neuronas)
    - Activación relu o swish, con Dropout y BatchNormalization
- **Capa de salida:** 1 neurona, activación sigmoid para clasificación binaria
- **Optimización:**
    - Adam con learning_rate optimizado
    - EarlyStopping
    - ReduceLROnPlateau
    - KerasTuner con Hyperband
- **Evaluación:** El uso de Embeddings en el conjunto de reseñas en otro idioma filtradas dio el mejor resultado para este modelo, convirtiéndolo en el segundo mejor modelo solo por detrás del ensamble.

La arquitectura fue elegida por su capacidad sobre tareas de texto representado como vectores. Se incorporaron capas de normalización y regularización para evitar overfitting, dadas las dimensiones elevadas de entrada.

Ensamble de modelos

Se generó un modelo final por votación mayoritaria combinando las predicciones de:

- Bayes Naive
- XGBoost(Embeddings)
- Red Neuronal(Embeddings)

Esto permitió mejorar la robustez general del sistema, compensando errores de los modelos individuales. **Obteniendo así el mejor modelo de todos para la competencia en Kaggle.**

Conclusiones Generales

El desarrollo del trabajo permitió abordar en profundidad un problema real de clasificación de texto, aplicando técnicas de procesamiento de lenguaje natural combinadas con modelos de machine learning.

Sobre los datos

- El dataset resultó ser balanceado (50% críticas positivas y 50% negativas), lo cual facilitó el entrenamiento sin necesidad de técnicas de balanceo.
- Se observaron críticas de distintas longitudes, redacciones informales, presencia de números y puntuación diversa, lo cual implicó aplicar una limpieza de texto cuidadosa.
- El análisis exploratorio fue clave para entender estas características, detectar outliers y definir decisiones de preprocesamiento.

Sobre el preprocesamiento

- La limpieza básica fue imprescindible para mejorar la calidad de los datos.
- El uso de TF-IDF con unigramas, bigramas, y un límite de features de 10.000, resultó ser una buena combinación de performance y eficiencia. Sin embargo el uso de Embeddings otorgó los modelos de mejores resultados.
- Se detectaron reviews en otro idioma (aproximadamente un 3%) en el conjunto de train y se decidió crear un dataset alterno ‘limpio’ sin ellas para realizar pruebas. Se utilizó Langdetect.
- La combinación de Embeddings y el dataset sin reviews en otro idioma aumentó ligeramente la performance solo con el modelo de redes neuronales.

Cuadro comparativo

Modelo	Velocidad	Complejidad	Comentarios
Naive Bayes	Alta	Baja	Útil como baseline rápido
Random Forest	Media	Media	El peor en rendimiento con tuning básico
XGBoost	Baja	Alta	De los mejores modelos con tuning y embeddings
Red Neuronal	Muy baja	Alta	El mejor modelo individual con tuning y embeddings. Mejor desempeño en test.
Ensamble	Muy alta	Baja	Mejor resultado en la competencia utilizando los mejores modelos

Sobre los modelos

- Se considera que el mejor modelo (individual) obtuvo métricas sólidas (F1-score > 0.90 en validación), lo cual habla de un buen desempeño en datos no vistos, aunque en el conjunto de prueba de la competencia el rendimiento bajó considerablemente (0.768). Esto indica que el modelo generaliza razonablemente, pero aún hay espacio para mejorar su robustez antes de implementarlo en un entorno productivo.

¿Cómo podrían mejorarse los resultados?

- Entrenar con un corpus más grande y diverso para ampliar el vocabulario del modelo
- Emplear modelos pre entrenados avanzados
- Ajustar hiper parámetros más finamente usando tiempo y más recursos computacionales

Tareas Realizadas

Integrante	Principales Tareas Realizadas	Promedio Semanal (hs)
Ignacio Carrera	Preprocesamiento y armado de modelos	4
Agustín Trombetta	Armado de modelos	4
Matias Etchegoyen	Preprocesamiento y ajustes de modelos	4
Braida Agustin	Armado de modelos	4