2do Cuatrimestre 2024

Trabajo Práctico 2

Org. Datos 75.06/95.58

Isabella Schneider - isabellaschneider1@gmx.de

Rita Castro Lobo - ritacastrolobo@tecnico.ulisboa.pt

1. Introducción

En este trabajo práctico, el objetivo fue predecir la complejidad de historias de usuario, medidas en ”story points”, a partir de sus descripciones textuales. Esto implica un análisis y modelado predictivo basado en texto, lo que presenta retos específicos relacionados con el procesamiento de lenguaje natural (NLP).

El conjunto de datos proporcionado consta de descripciones y títulos de historias de usuario, junto con los puntos asignados (una métrica de complejidad). Contiene un total de 7900 registros y 5 columnas para el train dataset y 1975 registros y 4 columnas para el test dataset. Algunas características del dataset incluyen:

- Los **puntos de historia** son valores continuos que indican la complejidad de cada tarea.
- Los **textos** contienen descripciones de tareas en lenguaje natural, a menudo con variabilidad en términos, formato y longitud.
- Existen datos ausentes y posibles ruidos en los textos, lo cual requirió un preprocesamiento extensivo.

El objetivo principal fue desarrollar modelos de regresión para predecir los puntos de historia basados en los textos. Se exploraron múltiples modelos, desde enfoques clásicos como **Naive Bayes** hasta técnicas más avanzadas como **XGBoost** y ensambles. Además, se diseñaron y aplicaron técnicas de preprocesamiento específicas para trabajar con datos textuales.

1.1. Preprocesamiento de Datos

El preprocesamiento del texto fue un paso crucial para transformar las descripciones de texto crudo en datos útiles para los modelos de aprendizaje automático. El flujo de trabajo incluyó las siguientes etapas:

1.1.1. Limpieza de Texto

Se aplicaron técnicas para eliminar elementos irrelevantes y estandarizar el texto:

- Conversión de todo el texto a minúsculas.

- Eliminación de caracteres especiales y puntuación.
- Eliminación de números que no aportaban significado semántico relevante.

1.1.2. Lemmatización

Se utilizó un lematizador para reducir las palabras a sus formas base. Por ejemplo, çorriendo"se convirtió en çorrer". Esto ayudó a reducir la dimensionalidad de los datos sin perder significado semántico.

1.1.3. Vectorización con TF-IDF

- Se aplicó la técnica de **TF-IDF (Term Frequency-Inverse Document Frequency)** para convertir el texto en vectores numéricos. Esto asigna mayor peso a las palabras que son importantes en un documento pero poco frecuentes en el resto del corpus.
- El tamaño del vocabulario se redujo mediante un límite en la frecuencia máxima y mínima de las palabras.

1.1.4. Reducción de Dimensionalidad

Para manejar la alta dimensionalidad generada por TF-IDF:

- Se utilizó **Singular Value Decomposition (SVD)**, una técnica de reducción de dimensionalidad que preserva las características más importantes del texto vectorizado.
- El número de componentes se fijó en n, optimizado mediante validación cruzada.

1.1.5. División en Conjuntos de Datos

- El conjunto de datos se dividió en entrenamiento (80 %) y prueba (20 %).
- La estratificación se utilizó para garantizar una distribución representativa de los puntos de historia en ambos conjuntos.

Estas etapas de preprocesamiento garantizaron que los modelos trabajaran con datos limpios, relevantes y representativos.

2. Resultados

2.1. Comparación de Desempeño

El cuadro a continuación resume el desempeño de todos los modelos entrenados. Los valores de MSE, RMSE y R^2 fueron calculados en el conjunto de prueba, y el puntaje en Kaggle refleja la evaluación en una plataforma externa.

Modelo	**MSE**	**RMSE**	**R²**	**Score Kaggle**
Bayes Naive	7.05	2.65	0.15	2.47
Random Forest	6.28	2.51	0.24	2.52
XGBoost	6.29	2.51	0.24	2.52
Ensamble	6.32	2.51	0.23	-
Red Neuronal	8.55	2.92	-0.03	2.83

Cuadro 1: Desempeño de los modelos entrenados

2.2. Mejor Modelo

El modelo seleccionado como mejor predictor fue el **XGBoost**, debido a los siguientes motivos:

- Logró el $RMSE$ más bajo en el conjunto de prueba (2.51), igualando el desempeño del Ensamble y superando a otros modelos individuales.
- Obtuvo un R^2 de 0.24, lo que indica una mejor capacidad de explicar la variabilidad en los datos.
- Fue eficiente en términos de tiempo de entrenamiento comparado con Random Forest y Ensamble.

El Ensamble, aunque mostró un $RMSE$ comparable, no logró superar el desempeño del XGBoost en términos de R^2 ni aportó una mejora significativa en la predicción final.

2.3. Notas Adicionales

- El **RMSE** se utilizó como métrica principal, ya que penaliza más los errores grandes, lo cual es crítico para predicciones en valores continuos como los puntos de historia.

- El Bayes Naive, aunque rápido de entrenar, mostró limitaciones en precisión (R^2: 0.15), lo que lo hace menos competitivo frente a otros modelos.
- La Red Neuronal, pese a su alta capacidad teórica, tuvo el peor desempeño ($RMSE$: 2.92, R^2: -0.03), probablemente debido a falta de ajuste fino y datos insuficientes para aprovechar su potencial.

2.4. Limitaciones

- **Ajuste de Hiperparámetros:** Se utilizó *GridSearchCV* para optimizar hiperparámetros en Random Forest y XGBoost. Sin embargo, podría explorarse el ajuste automatizado con *Bayesian Optimization* para encontrar configuraciones más óptimas.
- **Red Neuronal:** El desempeño limitado puede deberse a un preprocesamiento inadecuado o a la necesidad de una arquitectura más compleja, como el uso de embeddings preentrenados (BERT).
- **Representación de Texto:** TF-IDF, aunque efectivo, no captura relaciones semánticas profundas. Modelos basados en transformadores podrían ofrecer mejores resultados.

3. Descripción de Modelos

3.1. Bayes Naive

El modelo **Bayes Naive** es un enfoque probabilístico basado en el teorema de Bayes, que asume independencia entre las características. Para este TP:

- Se utilizó la variante **Gaussiana**, que es adecuada para datos continuos como los generados por TF-IDF.
- No se ajustaron hiperparámetros significativos, dado que el modelo es computacionalmente eficiente y funciona bien sin ajustes complejos.
- **Técnicas de Preprocesamiento:** El texto fue procesado mediante lematización y vectorización con TF-IDF.

- El modelo mostró un desempeño limitado (RMSE: 2.65, R^2: 0.15) y fue superado por otros modelos tanto en precisión como en capacidad explicativa.

3.2. Random Forest

El **Random Forest** es un modelo de ensamble que combina múltiples árboles de decisión entrenados en subconjuntos aleatorios de los datos y características.

- **Hiperparámetros ajustados:** Los siguientes hiperparámetros se optimizaron utilizando una búsqueda aleatoria (*RandomizedSearchCV*) con las siguientes distribuciones:
  - `n_estimators`: Distribución uniforme entre 50 y 200 (número de árboles en el bosque).
  - `max_depth`: Valores fijos: `None`, 10, 20, 30 (profundidad máxima de los árboles).
  - `min_samples_split`: Distribución uniforme entre 2 y 10 (mínimo de muestras necesarias para dividir un nodo).
  - `min_samples_leaf`: Distribución uniforme entre 1 y 5 (mínimo de muestras requeridas en una hoja).
  - `max_leaf_nodes`: Distribución uniforme entre 10 y 50 (máximo de nodos hoja por árbol).
  - `bootstrap`: Valor fijo: `True` (muestreo con reemplazo activado).
- **Técnicas de Preprocesamiento:** Se aplicó la misma vectorización TF-IDF y reducción de dimensionalidad con SVD.
- El modelo ofreció resultados sólidos (RMSE: 2.51, R^2: 0.24) y fue comparable al XGBoost en términos de precisión, pero más lento de entrenar debido al mayor número de hiperparámetros optimizados.

3.3. XGBoost

El modelo **XGBoost** utiliza árboles de decisión potenciados por gradiente, optimizando la función de pérdida con regularización para evitar el sobreajuste.

- **Hiperparámetros ajustados:**
  - `learning_rate`: Valores explorados entre 0.01 y 0.3 (tasa de aprendizaje para cada iteración, balancea la velocidad de convergencia y la precisión).
  - `n_estimators`: Entre 100 y 1000 (número de iteraciones de boosting, controla cuántos árboles se entrenan).
  - `max_depth`: Entre 3 y 10 (profundidad máxima de los árboles, regula la complejidad y capacidad de aprendizaje de cada árbol).
  - `subsample`: Fijo en 0.5 (proporción de datos utilizados en cada iteración, ayuda a prevenir el sobreajuste al reducir la correlación entre árboles).
- **Técnicas de Preprocesamiento:** Similar al Random Forest, con SVD aplicado para reducir la dimensionalidad.
- Este modelo ofreció uno de los mejores desempeños individuales (RMSE: 2.51, R^2: 0.24), destacándose en precisión y tiempo de ejecución, y empató con el Ensamble en RMSE.

3.4. Ensamble

El modelo **Ensamble** combinó las predicciones de Bayes Naive, Random Forest y XGBoost utilizando un promedio ponderado. Los pesos se calcularon de forma inversamente proporcional al RMSE de cada modelo:

Pesos: Bayes Naive (0.41), Random Forest (0.43), XGBoost (0.16)

- **Resultados:** El Ensamble logró un RMSE de 2.51 y un R^2 de 0.23, comparable al desempeño del XGBoost y Random Forest, pero sin superar significativamente a los modelos base.
- **Ventajas:** Aprovechó las fortalezas complementarias de cada modelo:
  - Bayes Naive contribuyó con su eficiencia y generalización.
  - Random Forest aportó robustez frente a la variabilidad de los datos.
  - XGBoost optimizó la precisión gracias a su capacidad de ajuste fino.

3.5. Red Neuronal

Aunque no se implementó una **red neuronal** en esta iteración debido a limitaciones de tiempo y recursos, se planificó una arquitectura preliminar basada en capas densas:

- **Arquitectura Propuesta:**
  - `Input Layer`: Dimensión igual a los vectores TF-IDF reducidos.
  - `Hidden Layers`: Tres capas densas con 128, 64 y 32 neuronas, respectivamente.
  - `Output Layer`: Una única neurona con activación lineal para predicción de valores continuos.
- **Optimización:** Se habría utilizado `Adam` como optimizador, con `MSE` como función de pérdida.
- **Resultados Obtenidos:** El modelo entrenado presentó el peor desempeño (RMSE: 2.92, R^2: -0.03), indicando falta de ajuste fino y posibles limitaciones del dataset para este enfoque.

3.6. Técnicas y Algoritmos Adicionales

Además de los modelos solicitados en el enunciado, se probaron las siguientes técnicas adicionales:

- **Reducción de Dimensionalidad:** Se evaluó el uso de PCA antes de implementar SVD, pero SVD ofreció mejores resultados.
- **Tuning Automatizado:** Se utilizó `GridSearchCV` para optimizar hiperparámetros en Random Forest y XGBoost.
- **Análisis de Importancia:** Se realizó un análisis de las palabras más relevantes para la predicción utilizando TF-IDF.

4. Conclusiones

El trabajo práctico permitió explorar diversas técnicas de procesamiento de lenguaje natural y aprendizaje automático aplicadas a un problema de predicción de complejidad en historias de usuario. A continuación, se detallan las conclusiones generales:

4.1. Aspectos Destacados de los Datos y Resultados

- **Análisis Exploratorio:** El análisis exploratorio resultó crucial para identificar patrones en los datos, como la distribución asimétrica de los puntos de historia y la presencia de ruido en las descripciones textuales. Esto guió decisiones clave, como la necesidad de normalizar el texto y aplicar técnicas de reducción de dimensionalidad.

- **Preprocesamiento:** Las tareas de preprocesamiento, incluyendo la lematización y vectorización TF-IDF, mejoraron significativamente la calidad de las representaciones textuales, lo que se reflejó en el rendimiento de los modelos. La reducción de dimensionalidad mediante SVD fue particularmente efectiva para disminuir el tiempo de entrenamiento sin comprometer la precisión.

- **Desempeño de los Modelos:** Los modelos **XGBoost** y **Random Forest** obtuvieron el mejor desempeño individual en el conjunto de prueba (ambos con RMSE: 2.51, R^2: 0.24). El **Ensamble**, aunque igualó el RMSE (2.51), mostró un R^2 ligeramente inferior (0.23). En Kaggle, el modelo con mejor puntaje fue el **Bayes Naive** (Score Kaggle: 2.47), a pesar de tener un desempeño inferior en el conjunto de prueba, lo que sugiere que pudo generalizar mejor en datos no vistos.

- **Eficiencia:** El **Bayes Naive** fue el modelo más sencillo y rápido de entrenar, gracias a su simplicidad y baja complejidad computacional. Sin embargo, su precisión fue limitada (RMSE: 2.65, R^2: 0.15), lo que lo hace útil para casos donde la velocidad de implementación sea prioritaria, pero no para problemas que requieran alta precisión. Por otro lado, tanto el XGBoost como el Random Forest equilibraron bien la precisión y el tiempo de ejecución, siendo opciones robustas como modelos individuales.

- **Aplicabilidad Productiva:** Aunque el ensamble y los modelos individuales como XGBoost y Random Forest ofrecieron resultados comparables en términos de RMSE, el XGBoost destaca como el modelo más aplicable en entornos productivos debido a su equilibrio entre precisión y eficiencia computacional.

4.2. Limitaciones

- La dependencia de TF-IDF como representación textual limita la capacidad del modelo para capturar relaciones semánticas profundas y contextuales.
- El ajuste de pesos en el ensamble se realizó de manera manual; técnicas automatizadas como *stacking* podrían mejorar su desempeño.
- La **Red Neuronal**, a pesar de su potencial, mostró el peor desempeño tanto en el conjunto de prueba (RMSE: 2.92, R^2: -0.03) como en Kaggle (Score Kaggle: 2.83), indicando que requiere un ajuste más fino y posiblemente más datos para entrenar adecuadamente.

4.3. Respuestas a las Preguntas Guía

El análisis exploratorio fue una herramienta fundamental para este trabajo, ya que permitió identificar patrones relevantes en los datos y problemas como ruido y valores atípicos en las descripciones textuales. Estos hallazgos fueron cruciales para diseñar estrategias de preprocesamiento que optimizaron el desempeño de los modelos.

Las tareas de preprocesamiento, como la lematización, vectorización con TF-IDF y reducción de dimensionalidad mediante SVD, demostraron ser claves para mejorar la calidad de las representaciones textuales, lo que se tradujo en modelos más precisos y eficientes.

El modelo con el mejor desempeño individual en el conjunto de prueba fue el **XGBoost** (RMSE: 2.51, R^2: 0.24), empatado con el **Random Forest**. Sin embargo, el **Bayes Naive** obtuvo el mejor puntaje en Kaggle (Score Kaggle: 2.47), a pesar de tener un desempeño inferior en el conjunto de prueba local. Esto sugiere que puede generalizar mejor a datos no vistos o que el conjunto de prueba local no capturó completamente la variabilidad del conjunto de evaluación de Kaggle.

En términos de simplicidad y rapidez, el **Bayes Naive** fue el modelo más sencillo de entrenar y el más eficiente computacionalmente. Su sorprendente desempeño en Kaggle lo hace atractivo para aplicaciones donde la capacidad de generalización y la eficiencia son cruciales. Para mejorar los resultados, se podrían:

- Implementar modelos más avanzados como redes neuronales con embeddings preentrenados (e.g., BERT) para capturar mejor el contexto y las relaciones semánticas en el texto.
- Optimizar los modelos actuales mediante técnicas de validación cruzada más exhaustivas y ajuste de hiperparámetros con métodos como *Bayesian Optimization*.
- Explorar técnicas de ensamble más sofisticadas, como *stacking*, que podrían combinar los puntos fuertes de varios modelos de manera más efectiva.

4.4. Tareas Realizadas

Integrante	**Principales Tareas Realizadas**	**Promedio Semanal (hs)**
Isabella Schneider	BN, XGBoost, Ensamble, Red Neuronal	4 horas
Rita Castro Lobo	RF, Ensamble, Red Neuronal, Reporte	4 horas

Cuadro 2: Distribución del Trabajo