Trabajo Práctico N°2 - Grupo 03

Introducción

Este trabajo se centra en poder predecir, mediante un texto en lenguaje natural, los valores de story points usando distintos modelos de machine learning.

El dataset contiene 1975 filas de 4 columnas, donde la columna 'title' de tipo texto contiene el título del user story, la columna 'description' , del mismo tipo, detalla el problema del user point, un columna 'project' que muestra a qué proyecto pertenece el caso y finalmente el 'storypoint' de tipo numérico que le da un valor o dificultad al problema planteado.

Analizando el dataset, no se encuentra presencia de valores nulos, la distribución de story points, relación longitud-descripción y longitud-título no son uniformes. Tampoco hay una distribución pareja entre storypoints por cada proyecto.

Utilizamos one hot encoding para la columna de 'project', decidimos concatenar el título y la descripción en una sola columna para poder analizarla entera posteriormente.

A lo largo de todo el trabajo se utilizaron técnicas de limpieza de lenguaje natural para poder mejorar las predicciones de los modelos. Pasamos todas las letras a minúscula, eliminamos signos de puntuación, números, caracteres no alfabéticos y stopwords, filtramos palabras de baja frecuencia, lematizamos el texto y finalmente usamos distintos tokenizadores.

Hipótesis y supuestos del grupo:

- Cuanto mayor y más complejo sea el preprocesamiento, mejor será la precisión de los modelos de predicción de story points.
- Si los story points los pone una persona, un modelo computacional tendrá mucha dificultad para predecir con tanta certeza ese valor.
- La red neuronal permite predecir el story point de una tarea con menor RMSE en comparación con los otros modelos implementados.
- El modelo de ensamble, dará los mejores resultados, debido a que combina otros modelos.

Cuadro de Resultados

Modelo	MSE	RMSE	Kaggle
Bayes Naive	7.1664	2.6770	3.0254
Random Forest	5.8843	2.4257	2.8366
XgBoost	6.019	2.453	3.0388
Red Neuronal	6.1358	2.4771	2.82991
Ensamble Stacking	**5.5429**	**2.3543**	**2.7707**
Ensamble Voting	5.8377	2.4161	2.7721

El mejor predictor obtenido es el ensamble de KNeighborsRegressor - RandomForestRegressor - XGBoostRegressor - Red Neuronal.

Descripción de Pre-procesamiento de Datos

Decidimos unir las columnas 'title' y 'description' en una sola, llamada 'text'. Luego, sobre 'text' aplicamos diversas técnicas de preprocesamiento generando nuevas columnas:

- **'Clean_text'**: se aplica sobre 'text', la función *clean_text*, que limpia un texto convirtiéndolo a minúsculas, eliminando caracteres no alfabéticos y las stopwords en inglés, para dejar solo palabras relevantes
- **'Lemmatized_text'**: se aplica sobre 'text', la función *clean_text_lemmatizer,* limpia un texto convirtiéndolo a minúsculas, eliminando caracteres no alfabéticos, filtrando stopwords y de longitud menor o igual a 2, para luego lematizarlas (proceso de reducir una palabra a su forma base o raíz).
- **'Clean_filtered_text'**: sobre 'clean_text' se aplica la función *filter_by_min_freq,* para mantener solo las palabras que aparecen con una frecuencia mínima especificada (min_freq), en este caso se eligió una frecuencia mínima de 5.
- **'Filtered_text'**: mismo que el anterior, pero sobre 'lemmatized_text'

Descripción de Modelos

Bayes Naïve

En un principio probamos utilizando ***Bag of Words***: convierte texto en un vector numérico basado en la frecuencia de las palabras, ignorando el orden y la gramática. Para analizar otra opción y comparar resultados, luego utilizamos ***TfidfVectorizer***: transforma texto en un vector numérico ponderando las palabras según su frecuencia en un documento y su rareza en el conjunto de documentos, destacando las más relevantes. Lo limitamos a las 2000 palabras más relevantes del user story y consideramos unigramas, bigramas y trigramas.

Se utilizaron los siguientes hiperparámetros con **Grid Search** y cross validation (10 partes) para optimizar el modelo:

- **Alpha**: parámetro de suavizado para evitar probabilidades nulas.
- **Fit_prior**: utilizado para ajustar el balance entre las clases.

El mejor conjunto de hiperparámetros obtenido fue: **Alpha**: 20.0 y **Fit_prior**: True

Resultados según Pre-procesamiento y Vectorización

Preprocesamiento	Vectorización	Error Cuadrático Medio (MSE)	Raíz del Error Cuadrático Medio (RMSE)
Sin pre - procesamiento, columna 'text'	BoW	9.918	3.149
	TfidfV	7.301	2.702
'clean_text'	BoW	8.549	2.923
	TfidfV	**7.168**	**2.677**
'lemmatized_tex'	BoW	8.727	2.954
	TfidfV	7.209	2.685
'clean_filtered_text'	BoW	8.570	2.927
	TfidfV	**7.166**	**2.677**

Como podemos ver, el mejor resultado se obtuvo con el pre-procesamiento aplicado en **'clean_filtered_text'** y con **TfidfVectorizer**. Investigando, determinamos que Bayes Naive asume una independencia condicional entre las características. Por lo que, la lematización puede crear características demasiado generales (por ejemplo, "running" y "run" se agrupan en una sola palabra), lo cual puede reducir la especificidad, por lo que en este caso no fue útil, pero podría ser más efectiva en otros modelos. Además la eliminación de palabras infrecuentes redujo el ruido, y mejoró las métricas.

Random Forest

En este modelo utilizamos ***TfidfVectorizer*** para vectorizar el texto. Dado que intentamos predecir mediante regresión, usamos *RandomForestRegressor*.

Se usaron los siguientes hiperparametros con **Grid Search** y cross validation (10 partes) para optimizar el modelo:

- **'n_estimators':** número de árboles
- **'Max_depth':** profundidad máxima de los árboles
- **'min_samples_split'**: mínima muestra para dividir un nodo
- **'min_samples_leaf':** muestra mínima por hoja
- **'max_features':** máxima cantidad de características por división
- **'oob_score':** para permitir usar oob para estimar

El mejor conjunto de hiperparámetros obtenido fue: **'max_depth' :** None - **'max_features':** 'sqrt' - **'min_samples_leaf':** 1 - **'min_samples_split':** 5 - **'n_estimators':** 200 - **'oob_score':** True

En cuanto al pre-procesamiento, dado que este modelo y el siguiente, tomaron más tiempo de ejecución, investigamos y nos encontramos con que usar texto lematizado es generalmente mejor para modelos como Random Forest y XGBoost, ya que reduce la dimensionalidad y el ruido, permitiendo que los modelos se concentren en patrones más relevantes y robustos en los datos.

Por lo tanto nos centramos en las columnas 'lemmatized_text' y 'filtered_text'.

Resultados según Pre-procesamiento y Conjunto de Datos

Preprocesamiento	Conjunto	Error Cuadrático Medio (MSE)	Raíz del Error Cuadrático Medio (RMSE)
'lemmatized_text'	Entrenamiento	1.719	1.311
	Validación	5.884	2.425
'filtered_text'	Entrenamiento	1.704	1.305
	Validación	5.939	2.437

El resultado obtenido sugiere que el filtrado realizado en filtered_text pudo haber eliminado características que eran útiles para el modelo. La lematización en sí ayuda a reducir la dimensionalidad de forma controlada, pero al combinarla con un filtrado de palabras basado en frecuencia, puede que se pierdan palabras clave que interactúan con otras características.

XGBoost

Para el modelo XGBoost, utilizamos ***TfidfVectorizer*** y desde un principio optamos por los pre-procesamientos de 'lemmatized_text' y 'filtered_text'

Se usaron los siguientes hiperparametros con Grid Search y validación cruzada (10 pliegues) para optimizar el modelo:

- **learning_rate:** Controla la rapidez con que el modelo ajusta los pesos.
- **max_depth**: Limita la profundidad de cada árbol, evitando sobreajuste y mejorando la generalización.
- **n_estimators**: Define el número de árboles en el modelo, aumentando la precisión a medida que se incrementa, pero también el tiempo de entrenamiento.
- **subsample**: Es la fracción de muestras utilizadas para entrenar cada árbol.

El mejor conjunto de hiperparámetros obtenido fue: **'learning_rate'**: 0,1, **'max_depth'**: 7, '**n_estimators**': 200 , **'subsample'**: 0,8

Resultados según Pre-procesamiento y Conjunto de Datos

Preprocesamiento	Conjunto	Error Cuadrático Medio (MSE)	Raíz del Error Cuadrático Medio (RMSE)
'lemmatized_text'	Entrenamiento	1.997	1.413
	Validación	6.020	2.453
'filtered_text'	Entrenamiento	3.068	1.751
	Validación	**6.019**	**2.453**

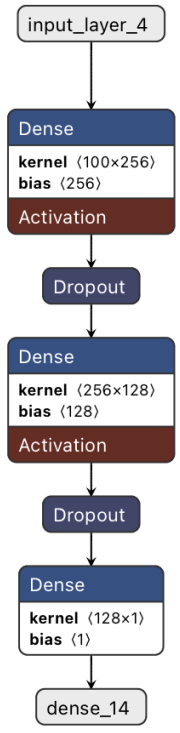
Aunque 'lemmatized_text' mostró un mejor rendimiento en el entrenamiento, el modelo con 'filtered_text' no fue significativamente peor en la validación y podría estar evitando el sobreajuste mejor que 'lemmatized_text'. La diferencia en la validación es mínima, lo que sugiere que ambas transformaciones de texto ofrecen resultados comparables en cuanto a la capacidad de generalización del modelo.

Red Neuronal

Se utiliza nuevamente *TfidVectorizer* para transformar el texto en vectores numéricos. Además, probamos implementando *Word2Vec,* para comparar.

Se usa KerasRegressor donde se define la siguiente arquitectura:

- **Capa de entrada:** la cantidad de entradas depende del tamaño de los vectores de entrada generados.
- **Primera capa densa:** Tiene 256 neuronas, utiliza relu como función de activación, se regulariza con L2 y dropout para evitar sobreajuste.
- **Segunda capa densa:** Tiene 128 neuronas (la mitad que la anterior), nuevamente usa relu como activación y L2 con dropout para evitar sobreajuste.
- **Capa de salida:** Una sola neurona con función de activación lineal.



Se usa para optimizar la red: **Adam**.

Utilizamos https://netron.app/ para visualizar el modelo entrenado:

Se utilizaron los siguientes hiperparámetros con **GridSearch** para optimizar el modelo:

- 'learning_rate': Tasa de aprendizaje
- 'dropout_rate': porcentaje de neuronas desconectadas
- 'l2_rate': regularización para evitar sobreajuste
- 'units': número de neuronas en la primer capa
- 'batch_size': cantidad de muestras procesadas por iteración

Además, se emplearon los siguientes **callbacks** para mejorar el entrenamiento:

- **EarlyStopping:** para detener el modelo cuando ya no mejora.
- **ReduceLROnPlateau:** reduce la tasa de aprendizaje cuando la métrica de validación no mejora.
- **ModelCheckpoint:** guarda el mejor modelo encontrado en disco.

El mejor modelo encontrado utiliza los siguientes hiperparámetros: **'batch_size':** 64 - **'dropout_rate':** 0.3 - **'l2_rate':** 0.1 - **'learning_rate':** 0.001 - **'units':** 128

Resultados según Pre-procesamiento y Vectorización

Preprocesamiento	Vectorización	Error Cuadrático Medio (MSE)	Raíz del Error Cuadrático Medio (RMSE)
Sin pre - procesamiento, columna 'text'	TfidfV	6.220	2.494
	Word2Vec	-	-
'clean_text'	**TfidfV**	**6.175**	**2.485**
	Word2Vec	7.479	2.734
'lemmatized_tex'	TfidfV	6.192	2.488
	Word2Vec	7.438	2.727
'clean_fltered_tex	TfidfV	6.182	2.486
	Word2Vec	7.316	2.704
'filtered_text'	TfidfV	6.199	2.489

	Word2Vec	7.162	2.676

Las redes neuronales tienen la capacidad de aprender representaciones complejas y capturar dependencias no lineales entre las características. Al simplificar el texto mediante preprocesamiento, se reduce la cantidad de datos que la red puede usar para aprender estos patrones complejos. Por lo tanto, el modelo podría desempeñarse mejor con datos menos procesados donde tiene más contexto para analizar. El MSE y RMSE de TF-IDF en entrenamiento y prueba son cercanos, lo sugiere que el modelo está generalizando mejor en comparación con Word2Vec.

Sugiere que una representación más explícita de la importancia de las palabras (como la que ofrece TF-IDF) fue más adecuada para el modelo de red neuronal en este caso. TF-IDF probablemente capturó de manera más precisa las características necesarias para la predicción de storypoint.

Ensamble

Ensamble Voting: Random Forest - RidgeRegressor - XGboost

El ensamble es del tipo voting combinando Random Forest- RidgeRegressor y XGboost, tres modelos de regresión para mejorar la precisión de las predicciones:

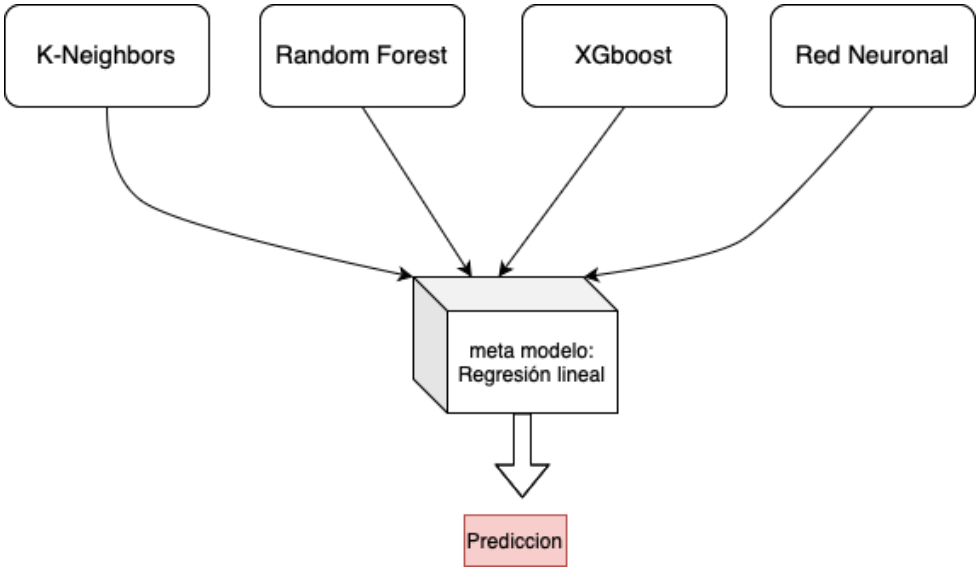
- **RandomForestRegressor:** Un modelo de bosque aleatorio con 200 árboles, que ajusta parámetros como la profundidad máxima y el número mínimo de muestras por hoja.
- **XGBRegressor:** Un modelo de XGBoost con parámetros optimizados como la tasa de aprendizaje y el número de estimadores.
- **Ridge:** Un modelo de regresión lineal con regularización L2 (Ridge), para manejar posibles sobreajustes.

El VotingRegressor combina estos tres modelos utilizando un enfoque de **votación ponderada**.

Ensamble Stacking: KNeighborsRegressor - RandomForestRegressor - XGBoostRegressor - Red Neuronal

El ensamble es del tipo stacking combinando un K-Neighbors, Random Forest, XGboost y una Red Neuronal.
El k-Neighbors usa 5 vecinos para predecir el valor. En cuanto a random forest, xgboost y la red neuronal utilizan los mejores hiperparametros de los modelos anteriores.

Se utiliza como meta modelo una regresión lineal, este se entrena para aprender a combinar lo modelos antes mencionados para saber cómo combinarlos para conseguir las mejores predicciones.



Comparación entre Modelos de Ensamble

Modelo	Conjunto	Error Cuadrático Medio (MSE)	Raíz del Error Cuadrático Medio (RMSE)
Voting	Entrenamiento	2.873	1.695
	Validación	5.837	2.416
Stacking	Entrenamiento	1.7442	1.3207

	Validación	**5.542**	**2.354**

Conclusiones generales

El análisis exploratorio fue esencial para comprender la distribución de los storypoints y la estructura del texto, lo que nos permitió identificar los pre-procesamientos más adecuados para mejorar el rendimiento de los modelos.

Las tareas de pre - procesamiento, como la lematización, eliminación de stopwords y normalización del texto, resultaron esenciales para mejorar la performance de los modelos. Vimos que el mejor pre-procesamiento variaba según el modelo utilizado, para algunos fue mejor uno básico y para otros cuanto más preprocesado mejor resultados.

El modelo de ensamble fue el que mostró el mejor rendimiento en el conjunto de prueba (TEST), con el menor RMSE. La combinación de K-Neighbors, Random Forest, XGBoost y la red neuronal permitió lograr un muy buen desempeño. En Kaggle, aunque Random Forest y XGBoost también obtuvieron buenos resultados, el modelo de ensamble fue superador.

Bayes Naïve fue el más sencillo y rápido de entrenar, aunque su rendimiento fue inferior, lo que limita su utilidad práctica en comparación con modelos más avanzados.

Creemos que el modelo de ensamble es viable para un uso productivo, aunque su complejidad y tiempo de entrenamiento deben ser considerados.

Para obtener mejores resultados, nos parece necesario un conjunto de datos más completo y equilibrado. Tener un dataset con una distribución más amplia y representativa de los diferentes valores de storypoints permitiría que los modelos aprendieran de una mayor variedad de ejemplos, lo que podría mejorar su capacidad de generalización y precisión en la predicción. También, se podrían haber aplicado técnicas de muestreo, como el sobremuestreo o el submuestreo.

Finalmente, entonces, analizando nuestras hipótesis iniciales, vemos:

- *Cuanto mayor y más complejo sea el pre-procesamiento, mejorar será la precisión de los modelos de predicción de storypoints.*

Un mayor pre-procesamiento no implica que los resultados sean mejores, de hecho, en algunos modelos dio mejores métricas, un pre-procesamiento más básico.

- *Si los story points los pone una persona, un modelo computacional tendrá mucha dificultad para predecir con tanta certeza ese valor.*

Puede que sea correcto, ya que, tiene cierta subjetividad, pero los modelos implementados se acercaron bastante al resultado correcto, y podrían ser mejorados.

- *La red neuronal permite predecir el storypoint de una tarea con menor RMSE en comparación con los otros modelos implementados.*

En este caso puntual, y con nuestra implementación, vemos que no se cumplió. No siempre el modelo más complejo, es el más adecuado para obtener los mejores resultados.

- *El modelo de ensamble, dará los mejores resultados, debido a que combina otros modelos.*

  Este supuesto, efectivamente se cumplió.

Extra

En el Análisis Exploratorio nos pareció interesante analizar el impacto del largo del título y de la descripción con el valor de storypoint, y además darle importancia a la columna “proyect”, que quizá también podía tener cierta relación.

Los resultados del modelo **Bayes Naive**, utilizando las nuevas columnas nos dieron:

- MSE entrenamiento: 8.331
- MSE validación: 7.032
- RMSE entrenamiento: 2.886
- RMSE validación: 2.651

Este es superador a los obtenidos anteriormente en Bayes Naive, por lo que creemos es positivo sumar información extra al texto.

Lamentablemente no seguimos profundizando en este tema, por cuestiones de tiempo y dificultad, pero nos pareció interesante comentarlo y dejarlo tanto en el informe como en el colab.

Tiempo dedicado

Integrante	Tareas	Prom. Hs Semana
Francisco Manuel Zimbimbakis	Pre-procesamiento, Bayes Naïve, Xgboost, Red Neuronal, Informe	10
Celeste De Benedetto	Analisis exploratorio, pre-procesamiento, Bayes naïve, Random Forest, Xgboost, Red Neuronal, Ensamble, informe	13
Morena Sandroni	Pre-procesamiento, red neuronal, informe	7