TP2: Story Points - Grupo 9

Nombre	Padrón	Mail
Josefina Marianelli Keichian	106798	jmarianelli@fi.uba.ar
Kevin Alberto Vallejo	109975	kvallejop@fi.una.ar
Lucas Oshiro	107024	loshiro@fi.uba.ar
Brenda Aylas	99575	baylas@fi.uba.ar

Introducción:

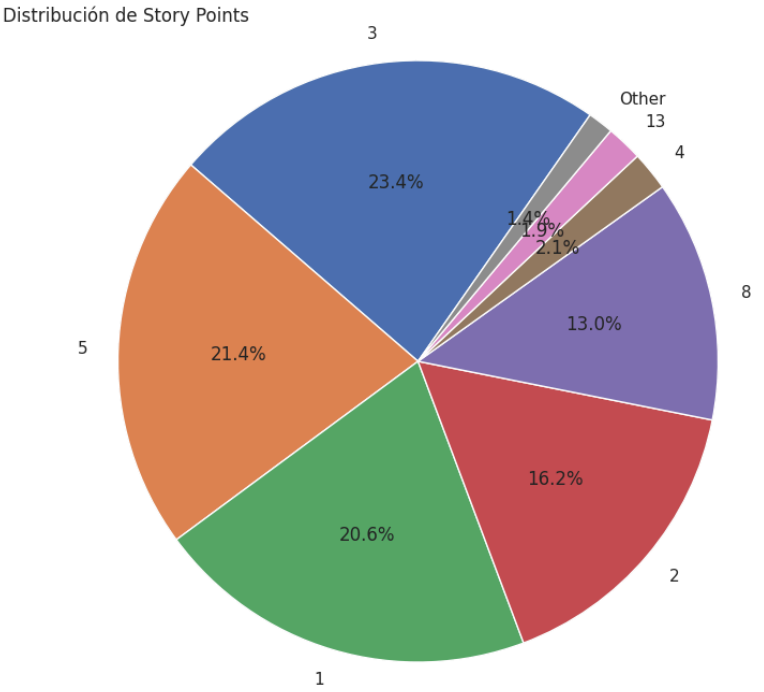
El presente trabajo práctico tiene como objetivo predecir a través de modelos el valor de story points asociados a casos de uso de diferentes proyectos. Los diferentes números posibles representan la complejidad de cada tarea. Como dataset para entrenar los diferentes modelos se tienen los valores mencionados: los casos de uso tienen una columna de título y otra de descripción, ambas en forma de texto.

El dataset utilizado para entrenar los modelos cuenta con 5 columnas y 7900 registros.

Sus diferentes columnas y tipo de datos son:

- ID: int
- Title: texto
- Description: texto
- Project: texto con un número al final que indica a cuál proyecto pertenece
- Story Point: int

Utilizando la función describe verificamos que no hay valores nulos en las columnas. En el dataset de entrenamiento se puede observar que hay 4 storypoints que conforman aproximadamente el 80% de los registros por lo que se intuye va a suceder algo similar con respecto a las proporciones cuando se predigan con nuestros modelos, tal como se puede ver en la siguiente figura:



El 80% de los registros están asociados a los storypoints de 3, 5, 1 y 2 en ese orden.

En la categoría de “Other” se encuentran los que tenían un porcentaje menor al 1% y estos fueron: 10, 12, 13, 14, 15, 16, 20, 21, 24, 32, 34, 40.

Algunos supuestos tomados:

- Manejo de los StoryPoints y la secuencia de Fibonacci: Observamos que los valores de StoryPoints seguían una secuencia basada en los números de Fibonacci. Inicialmente, realizamos pruebas eliminando los valores que no coincidían con esta secuencia, pero los resultados obtenidos en Kaggle no fueron satisfactorios. Después de evaluar el impacto, decidimos no eliminar los números fuera de la secuencia, ya que no generaban una mejora significativa en el rendimiento del modelo.
- Eliminación de StoryPoints con valor de conteo 1: Para algunos StoryPoints sólo existía 1 registro asociado a él (por ejemplo, 32, 34, 14, etc). Debido a las restricciones de las funciones de entrenamiento, que requieren al menos dos ocurrencias de cada clase, decidimos eliminar estos casos, ya que no cumplían con el criterio mínimo para ser incluidos en el modelo de manera efectiva.
- Exclusión de la columna "Proyecto": Durante el proceso de entrenamiento del modelo, se decidió no considerar la columna "Proyecto". Esto se debió a que, en el contexto del análisis, esta variable no aportaba información relevante para la predicción de los valores objetivos.

Cuadro de resultados:

Modelo	MSE	RMSE	R2	Kaggle
Bayes Naive	8.47	2.91	-1.39	2.99760
Random Forest	5.05	2.24	-1.41	2.88470

XGBoost	2.10	1.45	0.76	2.78301
Red Neuronal	4.82	2.19	0.27	2.82
Red Neuronal Bert	5.06	2.25	0.38 (Accuaracy)	3.05
Ensamble	6.07	2.46	0.25	3.02897

Descripción de Modelos:

Bayes Naive:

Se selecciona como mejor modelo predictor según los resultados obtenidos en Kaggle a uno que utiliza solamente la variable *title* del dataset, también se probaron implementaciones de solamente la columna description y por otra parte una concatenación de title + description pero aún así este fue el que nos dio mejores resultados numéricos.

Como preprocesamiento se utilizó una función de clean_text que eliminaba espacios y símbolos que no aportaban y eran recurrentes como //, ::, {}. A su vez, realizando un análisis de los textos incluidos, se usaron funciones para eliminar ruido como podían ser números hexadecimales, partes http o https de las páginas y transformaciones como por ejemplo: RecurrentTimeSearch a 3 palabras: recurrent time search. Se utilizó un tokenizer base que incluía las stopwords en inglés y transformar a minúsculas todas las palabras.

Random Forest:

Para este modelo se utilizó el mismo preprocesamiento pero en este caso se utilizó *description* como variable predictora. Se usó cross_val = 6 para verificar las métricas como así también tuning de hiperparámetros a través de RandomizedSearchCV con 17 iteraciones. Los mejores hiperparámetros encontrados fueron:

- 'n_estimators': 600
- 'min_samples_split': 15
- 'min_samples_leaf': 4
- 'max_features': 'sqrt'
- 'max_depth': 40

XGBoost:

1. **Preprocesamiento de los Datos**

- **Limpieza de texto**: En este caso, se utilizó una función simple de limpieza que elimina los espacios en blanco innecesarios de los textos (`clean_text`). Adicionalmente, se desarrolló una función personalizada de **tokenización** (`tokenizer_v3`), que convierte el texto a minúsculas, elimina caracteres no alfabéticos y realiza **lematización** utilizando el lematizador de **WordNet** de **NLTK**. Esto permitió reducir las palabras a su forma base, mejorando la calidad del modelo.

- **Vectorización de texto**: Se utilizó **CountVectorizer** de **scikit-learn** para convertir el texto en una matriz de características numéricas. Se configuraron unigramas y bigramas, y se estableció un umbral de frecuencia mínima de aparición (min_df=17) para filtrar n-gramas poco frecuentes. Además, se incluyó un conjunto de **stopwords** personalizadas para eliminar términos irrelevantes.

2. Entrenamiento y Optimización del Modelo

- **Modelo base (XGBRegressor)**: Se entrenó un modelo inicial utilizando **XGBoost Regressor** con los parámetros predeterminados: 1000 estimadores (n_estimators=1000), tasa de aprendizaje baja (learning_rate=0.01), y profundidad máxima de los árboles (max_depth=6).

- **Optimización de Hiper-parámetros**: Para mejorar el rendimiento del modelo, se utilizó **RandomizedSearchCV**. Este método de búsqueda aleatoria explora una amplia gama de combinaciones de hiper-parámetros. En total, se realizaron **500 ajustes** (10 pliegues de validación cruzada para cada uno de los 50 candidatos evaluados). Este enfoque permitió encontrar la combinación de hiper-parámetros que optimizó el desempeño del modelo:

    - **Mejores hiperparámetros** encontrados:
        - subsample: 1.0
        - reg_lambda: 1
        - reg_alpha: 0
        - n_estimators: 1000
        - min_child_weight: 1
        - max_depth: 10
        - learning_rate: 0.01
        - gamma: 0
        - colsample_bytree: 0.7

3. Evaluación del Modelo

- Se evaluó el modelo utilizando **validación cruzada** con 10 pliegues (K-fold cross-validation), lo que asegura una estimación robusta del rendimiento y reduce el riesgo de sobreajuste.

- Las métricas empleadas fueron:

    - **R^2 (Coeficiente de Determinación)**: Mide la proporción de la variabilidad de la variable dependiente que es explicada por el modelo.
    - **MSE (Error Cuadrático Medio)**: Indica el promedio de los errores al cuadrado de las diferencias entre las predicciones y los valores reales.
    - **RMSE (Raíz del Error Cuadrático Medio)**: Es la raíz cuadrada del MSE y proporciona una métrica en las mismas unidades que los datos originales, lo que facilita la interpretación.

4. Predicción y Guardado del Modelo

- Tras entrenar el modelo con los mejores hiper-parámetros encontrados, se utilizó para hacer predicciones sobre un conjunto de prueba. Para garantizar la consistencia, los datos de prueba fueron transformados utilizando el mismo **CountVectorizer** que en el entrenamiento.

- El modelo final fue guardado en un archivo .joblib para su uso posterior, lo que permite cargarlo rápidamente sin necesidad de re-entrenarlo.

5. Aplicaciones y Resultados

Este modelo puede aplicarse en tareas de regresión con datos textuales, donde se requiere predecir valores continuos a partir de descripciones o textos. El enfoque de optimización con **RandomizedSearchCV** y el uso de **XGBoost** ha demostrado ser eficaz para predecir valores de manera precisa, incluso con datos complejos y de alta dimensionalidad.

Conclusión:

El modelo XGBoost, optimizado con RandomizedSearchCV, demostró un muy buen rendimiento en tareas de regresión con datos textuales. La búsqueda de hiperparámetros y la validación cruzada permitieron ajustar el modelo para obtener predicciones precisas y robustas, lo que lo convierte en una herramienta efectiva y escalable para problemas de análisis de texto.

Redes Neuronales

Se trabajaron en dos tipos de redes neuronales:

- Redes Neuronales Tradicionales
- Redes Neuronales Profundas.

Redes Neuronales Tradicionales:

Primero probamos con un modelo muy básico de una sola neurona y un optimizador SGD. Este primer modelo funciona muy parecido a una regresión lineal simple.
El preprocesamiento del texto se hace con TF-IDF

A partir de este modelo probamos más modelos aumentando las neuronas, capas, cambiando la función de activación. Y en algunos usando técnicas de regularización.

En particular, el mejor de estos modelos de redes neuronales tradicionales es el modelo 6 que cuenta con las siguientes características:

- Concatenación de columnas title y descrption.
- Preproceso de texto: tokenización, padding y truncamiento para no perder información relevante.
- tokenizador de tensorflow
- input_dim=10000
- output_dim=16
- input_length=100
- Capa densa de 16 unidad. (que utiliza ReLU para que el resultado sea no linea)
- Una capa de salida ya que sólo esperamos predecir una storypoint.
- Optimizador Adam. Lo recomendaron en clase, probamos otros y fue el mejor, combina lo mejor de SGD y RMSProp
- Entrena con 10 miserables épocas. Generaliza mejor. Entrenamos uno hasta con 100 épocas que daba rmse de 0.3 en train pero en test dio 8.2. Para elegir la cantidad apropiada de épocas después de entrenar con 100 nos fijamos en el historial, y aunque pareciera siempre mejorar, en test empeoraba.

De un modelo igual a este usamos una función para que sólo prediga storypoints del 1 al 13 fibonacci pero al subirlo en la competencia daba peor.

Redes Neuronales Profundas:

Se hizo un preprocesamiento diferente. Esto porque un 96% de los storypoints pertenecían a muy pocos números, y estos números eran fibonacci. Por lo que se toma la decisión de crear un modelo de redes

neuronales profundas de clasificación que esté enfocado en poder predecir bien los primeros cinco números de fibonnaci: 1, 2, 3, 5, 8.
Esto es porque los storypoints se suelden definir con una técnica llamada planning poker:
El Planning póker es una técnica para estimar usando story points en donde cada miembro del Development Team debe otorgar un estimado en story points al ítem que se esté tratando, usando una sucesión de Fibonacci

Se utiliza el modelo pre entrenado de Bert. Página:
https://huggingface.co/docs/transformers/model_doc/bert#overview
El cual es excelente para problemas de predicción con texto.
Es mejor que TF-IDF o embeddings básicos ya que está pre entrenado con muco texto y aprendió a comprender relaciones semánticas complejas.

Algunas características:

- tokenizador oficial de BERT, que divide texto en "tokens" subpalabras y maneja palabras desconocidas mediante un token especial (<UNK>).
- 768 neuronas en la capa de salida
- Dropout 0.5 evita sobreentrenamiento. Se apagan aleatoriamente el 50% de las neuronas.
- optimizador AdamW con learning rate bajo (2e-5)
- scheduler lineal
- 16 lotes
- 10 épocas

De las épocas no se probaron más porque por cada época el modelo se tardaba aproximadamente 1 hora. Por lo que entrenar un sólo modelo demoraba medio día. Este problema nos retrasó para saber si estábamos haciendo las cosas bien o probar arquitecturas distintas. Sumado a ello usamos 3 copias de la notebook para correr en paralelo distintos entrenamientos y quedarnos con los mejores.
En kaggle dio 3.05.

Mencionar si se probaron técnicas o algoritmos adicionales a las solicitadas en el enunciado del TP2:

- Prueba de Embeddings con XGBoost.
- Redes Neuronales Profundas Bert de Clasificación
- Algoritmos para que siempre prediga el número fibonacci más cercano.
- Se hace un “Ensamble a Mano” De 3 modelos. entrenados para tareas especificas:
    - Predecir menores o mayores a 8. Dado este resultado se usa uno de los siguientes dos.
    - Predecir entre 1 y 8
    - Predecir mayores a 8

Modelo Ensambles

Los modelos de Ensambles consisten en tener varios modelos base, donde cada uno realiza la predicción, y luego a partir de todas las predicciones, se encuentra una solución o predicción definitiva. Para este trabajo práctico elegimos armar un modelo de ensamble del tipo Stacking

Teniendo en cuenta esto, nosotros decidimos encontrar tres modelos bases y un meta modelo

Antes de empezar con el armado de los modelos, realizamos un preprocesamiento del dataset, donde dropeamos aquellos storypoints con una sola medición (ej: storypoint: 32,34,14,24); realizamos una limpieza de textos mediante la función clean_text; se realizó una vectorización de los elementos mediante la función

CountVectorizer, usando la función anteriormente mencionada clean_text, una función tokenizer y un array de stopwords llamada stoplist; y por último hicimos la división entre set de entrenamiento y de testeo

En primer lugar, decidimos usar como modelos bases algunos de los mejores modelos armados para este trabajo.Utilizamos el mejor modelo de XGBoost, de Random Forest y de Redes neuronales. Como meta modelo, elegimos una Regresión lineal simple. Para este modelo, obtuvimos las siguientes métricas:

- RMSE: 2.4637
- MSE: 6.07
- R2: 0.25

En principio fueron las mejores métricas obtenidas

En segundo lugar, usamos simples implementaciones de XGBoost y Random Forest, y una red neuronal simple. Al ser modelos muy simples, las métricas obtenidas no fueron muy buenas en relación al anterior modelo armado.

- RMSE: 2.82
- MSE: 8.00
- R2: 0.01

Luego, fuimos realizando distintas combinaciones de modelos bases y meta modelos, como por ejemplo, en vez de utilizar los mejores modelos de un tipo, fuimos utilizando algunos con peores métricas; en vez de utilizar como meta modelo una Regresion lineal, usamos el mejor de XGBoost, etc. Sin embargo, las métricas daban parecidas pero no mejores al primer modelo armado.

De esta forma, para el modelo de ensamblaje, nos quedamos con el modelo Stacking con XGBoost_v7, Random_Forest_v2, y Red_Neuronal_v6 como modelos base, y como meta modelo una Regresión lineal

Conclusión

Luego de realizar los armados y entrenamientos de modelos, obtuvimos que el mejor modelo es el XGBoost V7, ya que nos dio las mejores métricas, en nuestro collab, RMSE: 1.45 y en Kaggle, RMSE: 2.78301.

En todos los modelos, se realizaron similares tareas de preprocesamiento, lo cual ayudó a obtener mejores métricas. Además, la búsqueda de hiperparametros, a través de cross-validation, mejoraron mucho las métricas.

En cuanto a la dificultad del armado de los modelos, los más fáciles fueron Bayes Naive y Dummy, siendo los peores en cuanto a métricas. Luego los más difíciles fueron Redes Neuronales y Ensamble Stacking, siendo Redes Neuronales Bert, las más complicadas, obteniendo métricas decentes. Y XGBoost, en término medio en cuanto a la dificultad, obtuvo las mejores métricas.

Por lo tanto, tanto en dificultad del armado, la velocidad y las métricas obtenidas, XGBoost cumple con la gran mayoría.