CIENCIA DE DATOS

STORY POINTS - TP2 - 2024 C2

Alumnos:

Avalos, Camila Lucia

100862 - Benitez Potocheck, Tomás

Roldan Montes, Cristian Eduardo

104239 - Sprenger, Roberta

Curso:

Rodriguez, Juan Manuel - Organización de Datos 75.06/95.58

Corrector:

Azul Villanueva

Introducción

El presente trabajo, está orientado en la predicción del atributo "***storypoints***" de un conjunto de datos. Dicho atributo representa la complejidad de una tarea del set.

El set de datos con el que trabajaremos:

	id	title	description	project	storypoint
0	5660	Error enabling Appcelerator services during ap...	When creating the default app, I encountered t...	project8	3
1	9014	Create a maintenance branch	As a developer, I'd like to have a maintenance...	project6	5
2	4094	Service Activity Monitoring Backend integrated...	SAM API used by SAM GUI	project1	5
3	811	fs::enter(rootfs) does not work if 'rootfs' is...	I noticed this when I was testing the unified ...	project5	2
4	4459	transform processor with script option is broken	Creating the following stream throws exception...	project6	2

Set de entrenamiento:

train_set ↦ 7900 filas y 5 columnas

Por otro lado, el set de datos utilizado para el test de los modelos que aplicaremos:

	id	title	description	project
0	3433	Add Run > Tizen Emulator menu action in App an...	The action will create the launch shortcut for...	project8
1	106	Chrome & IE mis-behavior	On Wed, Aug 4, 2010 at 12:21 PM, Bryan Beecher...	project2
2	7182	Problems with Publishing routes (on release re...	I have a problem with publishing routes in Nex...	project1
3	8985	Redis sink: better handling of module options/...	Please see the discussion here: https://githu...	project6
4	2149	java0.log generated by the SAM	I found an issue on the TAC 5.2.1, a java0.log...	project1

Set de prueba:

test_set ↦ 1975 filas y 4 columns

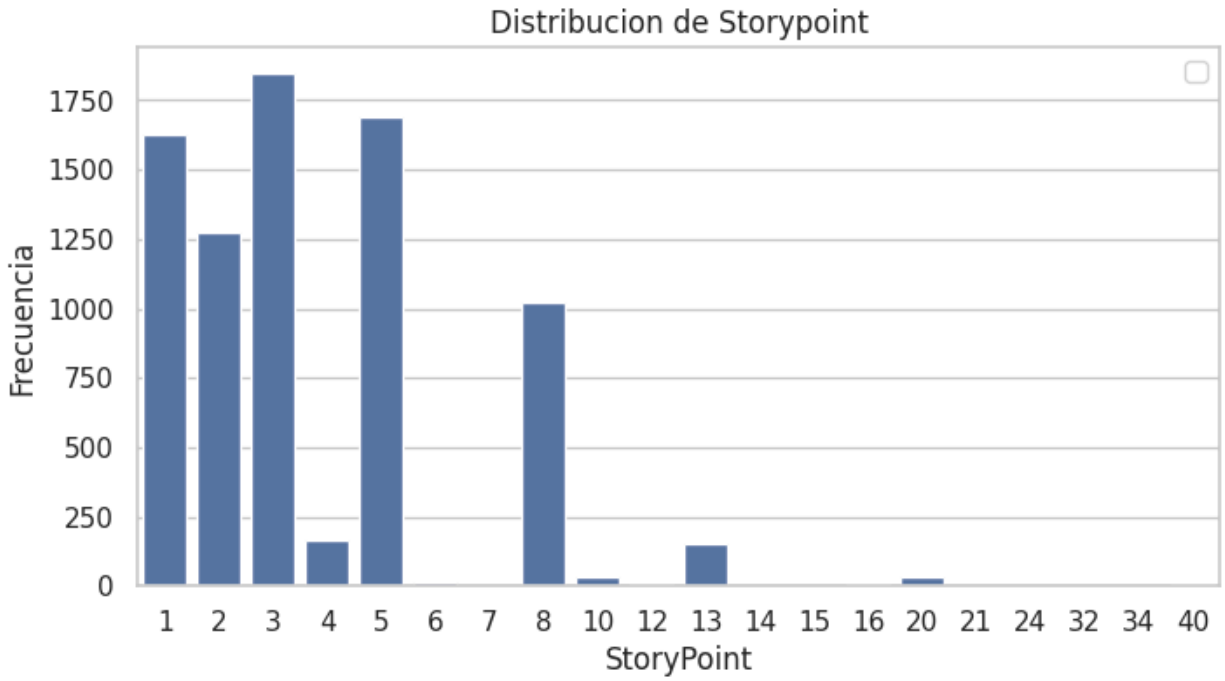
El objetivo es crear la columna ***storypoint*** para el set de testeo con las predicciones.

Cuadro de Resultados

Modelo	MSE	RMSE	MAE	Kaggle
Bayes Naive	7.877	2.807	1.648	2.9896
Random Forest	6.740	2.6	1.735	2.843
XG Boost	3.079	1.754	1.318	2.797
Red Neuronal				
Ensamble	4.398	2.097	1.488	2.994

PROCESAMIENTO

En nuestros análisis de procesamiento de lenguaje natural aplicamos diversas técnicas a los datos. Un paso común en todos nuestros modelos es la transformación de la columna `project` a una representación numérica, utilizando la librería LabelEncoder. Esta transformación es necesaria para que los algoritmos de aprendizaje automático puedan procesar los datos de manera efectiva.

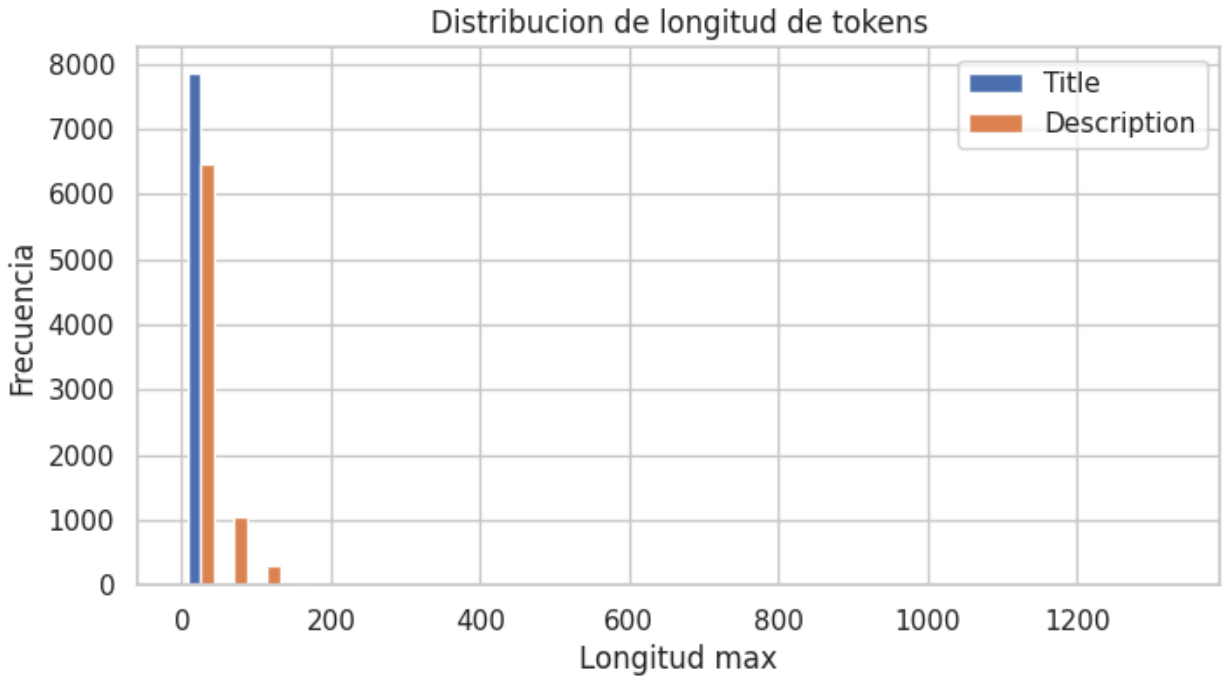


El set de datos se ve claramente desbalanceado, esto lo tomaremos en cuenta en el entrenamiento de los modelos, por ejemplo Random Forest o SVM ajustando los pesos y penalizando más los errores en los valores minoritarios. Esto incentiva al modelo a prestar más atención a las clases menos representadas.

Word tokenize:

Empleamos la librería **`NLTK`**, con la función word tokenize, aplicada a las columnas **`title`** y **`description`**. Esta función realiza la tokenización incluyendo signos de puntuación.

Para este procesamiento, retenemos los signos de puntuación asumiendo que estos pueden proveer información, luego veremos qué tanta información aportaron al compararlos con los distintos modelos a desarrollar.



ATRIBUTO	MAX	MIN	MEAN	MEDIAN	STD
DESCRIPTION	1331	3	28.25	14	33.25
TITLE	70	4	11.49	10	5.86

Para **`description`**, tras un filtrado de tokens con longitud superior a 14 caracteres, notamos que los tokens en cuestión corresponden a:

Categoría	**Descripción**
Archivos fuente	Nombres de archivos
Librerías	Librerías JAR
Clases	Clases en lenguajes
Funciones	Funciones usadas
Líneas de código	Números de línea

Esto es solo una breve descripción de los tokens encontrados filtrados con más de 14 caracteres, solo es un sample de la extensa cantidad que hay, lo que queremos ver es que estos tokens citados son textos bastante extensos, por ejemplo:

org.springframework.expression.spel.ast.CompoundExpression.getValueInternal
org.springframework.expression.spel.support.ReflectiveMethodExecutor.execute
org.springframework.data.mongodb.core.convert.MappingMongoConverter.read
'/tmp/._old_root_.XXXXXX'

La información que obtenemos en esos tokens, tienen un alto nivel de especificidad técnica, como nombres de rutas y nombres específicos de librerías, reflejando detalles específicos de las tecnologías involucradas. Sin embargo, su presencia repetida (en la mayoría de los casos) no necesariamente aporta información. También encontramos URLs y rutas de archivo que generalmente no aportan información sobre la dificultad de la tarea, estos datos podrían ser ruidosos.

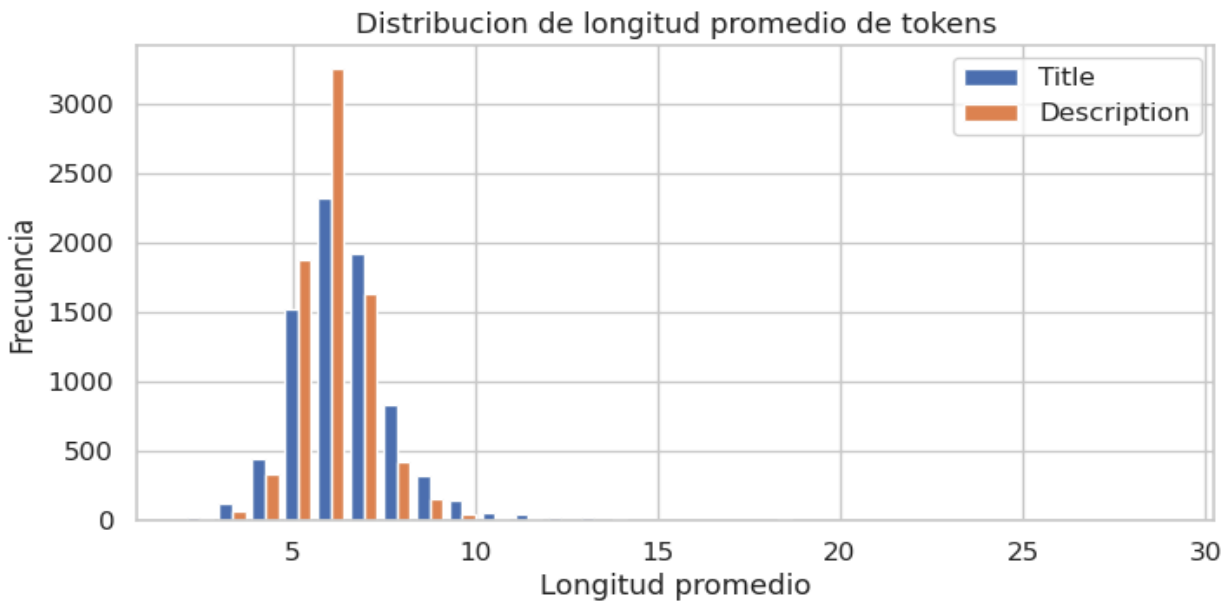
Por otro lado los mensajes de excepción como *SQLError.createSQLException* pueden ser útiles si ciertos errores recurrentes (por ejemplo, un error de conexión a la base de datos) se asocian con una dificultad específica.

En el cuadro anterior mostramos una tokenización básica, decidimos:

1. Eliminar expresiones de Ruta y Librerías:

   Con expresiones regulares identificamos rutas de archivo, versiones de librerías, URLs y otras expresiones largas, ya que no suelen agregar valor a la predicción.

2. Eliminar caracteres especiales.



Obtenemos tokens relativamente aceptables, la mayor distribución la encontramos entre longitudes de 3 a 7 caracteres por token.

TD - IDF + Lemmatization

TF-IDF (Term Frequency-Inverse Document Frequency) nos permite evaluar la importancia de una palabra en un documento. Al combinar la frecuencia de la palabra (TF) con la frecuencia inversa de documentos que contienen esa palabra (IDF), TF-IDF destaca las palabras que son más informativas en un contexto específico, ayudando a filtrar palabras comunes que no aportan valor a la predicción.

También empleamos esta técnica ya que reduce el ruido al desincentivar la consideración de palabras que aparecen en muchos documentos (las stop words). Esto permite que el modelo se enfoque en términos más específicos y relevantes para las características del problema abordado, en este caso, la estimación de story points.

Aplicaremos la técnica de lematización en el procesamiento, ya que preserva el significado semántico y reduce el ruido.

Bag of Words

Con la finalidad de comparar, procesamos el texto con BoW, ya que la frecuencia de las palabras las maneja de forma diferente. También aplicamos la lematización y eliminar stop words.

Modelos

XG BOOST

Preprocesamiento

Preprocesamos la columna `project`, convirtiéndola en una columna numérica de modo tal que el modelo de aprendizaje pueda procesar los datos.

También creamos una función que tokeniza y lemmetiza un texto pasado por parámetro y devuelve del mismo las 6 palabras más comunes que aparecen en él. Esto lo logramos con la librería de Spacy.

Aplicamos esta función tanto a la columna **`title`** como a la columna **`title`** para tener una bolsa de palabras con dos index.

El **`index`** sirve para poder representar el identificador de la palabra en las columnas y que el dato perteneciente a estas columnas represente la cantidad de veces que se encuentra la misma en el texto de las columnas previamente mencionadas.

Finalmente le retiramos las columnas que ya no necesitamos.

Modelo

Específicamente utilizamos una versión para regresión **`XGB Regressor`**. Creamos el modelo XGBoost y realizamos una búsqueda de hiperpárametros mediante **`RandomizedSearchCV`** y con 5 folds.

Esta decisión la tomamos en cuenta debido a que la gran cantidad de columnas e información del dataset hacían que la búsqueda y entrenamiento tengan un coste computacional alto y nos consumiera la ram disponible.

Usamos dos hiperparametros fijos

1. objective='reg:squarederror': Usamos esta opción, el error cuadrático medio (MSE).
2. eval_metric= rmse : Usamos este parámetro fijo debido a que rmse, era la métrica que queríamos mejorar.

Búsqueda de hiperparametros:

- 'max_depth': list(range(3, 15))
- 'learning_rate': [0.01, 0.1, 0.2]
- 'subsample': [0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
- 'n_estimators' : [50,100,150,200]

RMSE KAGGLE: 2.797

¿ Por qué obtuvimos este resultado ?

Tal como mencionamos en el preprocesamiento, los títulos y descripciones de las columnas en el data set, en muchos casos contenían lineas de codigo, mensajes de error, direcciones web entre otros, por lo que podríamos decir que tenemos un vocabulario un poco limitado, de esta forma Bag of Words pudo haber trabajado mejor.

Comparativa en cuanto a resultados con el set de entrenamiento

Cuando se entrenó al modelo y se obtuvieron las métricas respecto al set de entrenamiento. El RMSE dio un valor de 1.754 esto nos dice que el modelo puede haber generado un overfitting respecto a los datos de entrenamiento y esto generó que no pueda realizar bien las predicciones sobre nuevos datos.

WORD EMBEDDINGS

Para este modelo, usamos un preprocesamiento mucho más simple que en el caso anterior.

Para esto, tomamos las columnas **`title`** y **`description`**, las concatenamos y vectorizamos con RD - IDF.

También trabajamos con la columna **`project`**, la cual pasamos a una columna numérica mediante una discretización con LabelEncoder.

Utilizamos las herramientas de Keras se usan para preprocesar texto:

- **Tokenizer**: Para convertir el texto en secuencias numéricas basadas en un vocabulario.
- **pad_sequences**: Ajusta las secuencias generadas por Tokenizer para que todas tengan la misma longitud, rellenándolas con ceros o truncándolas.

Tokenizer:

- num_words=10000: Solo mantenemos las 10,000 palabras más frecuentes en los datos, con esto reducimos el espacio de búsqueda y evitamos trabajar con palabras muy raras, que suelen ser ruido.
- oov_token="<OOV>": Lo usamos para representar palabras que no están en el vocabulario.

Para este modelo utilizamos embeddings preentrenados (en este caso, GloVe) y un modelo de red neuronal con Keras para las predicciones

Cargamos un archivo de embeddings preentrenados llamado glove.6B.100d.txt, que contiene vectores de palabras ya aprendidos a partir de un gran corpus de texto.

Construcción de modelos con Keras

- model_relu_adam = Sequential()
- model_sigmoid_adam = Sequential()

Los dos modelos pasan por una capa de Embedding, donde transformamos los índices de palabras en vectores, establecemos una dimensión de cada vector en cien.

Capas Densas: Añadimos capas totalmente conectadas

  - 64 neuronas con activación ReLU (sigmoide para el otro modelo).
  - 32 neuronas con activación ReLU (sigmoide para el otro modelo).
  - 1 neurona (salida) sin activación, ya que se trata de un problema de regresión (predicción de valores continuos).

Modelo Sigmoide adam: MSE KAGGLE = 3.16454 / Modelo Relu Adam = 3.16454

BAYES NAIVE

Bayes Naive es un modelo muy simple y de relativamente pocos hiperparámetros que hace varias suposiciones sobre la estructura y relaciones en los datos que no suelen ser válidas, por lo tanto no esperamos los mejores resultados de este modelo. Sin embargo, sigue siendo un modelo fácil de entrenar con resultados generalmente buenos o no muy malos.

Preprocesamiento

Utilizamos el mismo tipo de preprocesamiento que para el XGBoost (tokenización, lematización de title y description, eliminación de stopwords). Utilizamos ambos tf-idf y bow para la vectorización del texto y nos quedamos con el mejor.

Modelo

Utilizamos la implementación `MultinomialNB` de sklearn. Optimizamos los hiperparámetros:

- 'alpha' : [0., 0.5, 1.0]
- 'fit_prior' : [True, False]

mediante una búsqueda randomizada, utilizando 5 folds estratificados para la validación cruzada.

RMSE Kaggle: 2.98957 (tf-idf)

obtuvimos el mejor resultado utilizando el esquema tf-idf para vectorizar. Como comparativa, estas fueron las métricas obtenidas en base al set de entrenamiento:

vectorización/métrica	mae	mse	rmse
tf-idf	1.648	7.877	2.807
bow	1.935	10.281	3.206

RED NEURONAL

Empleamos el modelo Word Embeddings, para el pre-procesamiento aplicamos una tokenización, usando las librerías ***tensorflow.keras.prepocessing.text*** y ***tensorflow.keras.prepocessing.sequence .***

En cuanto a las indicaciones para el tokenizador, aplicamos un número de 10.000 palabras más frecuentes en los textos, y usamos el parámetro *oov_token="<00V>* para agregar un token fuera del vocabulario.

Al igual que en los modelos anteriores, concatenamos los atributos "title" y "description" con la finalidad de crear un único conjunto de texto para tokenizar y entrenar con ambas columnas.

Luego, convertimos el texto anterior en una secuencia de números, pero asegurando que todas las secuencias tengan la misma longitud, si una secuencia es más corta, se rellena, si es más larga se trunca, para esto se pasaron los parámetros maxlen, padding y truncating.

Empleamos un archivo Embeddings, descargado de Glove, específicamente con el archivo *glove.6B.100d.txt.*

La idea es utilizar los embeddings pre entrenados de Glove, los cuales son vectores que representan el significado de las palabras en un espacio vectorial.

Modelo

Usamos el optimizador "ADAM" ya que es el considerado en la actualidad como el mejor, se adapta a la tasa de aprendizaje y es el que converge con más velocidad.

Entrenamientos:

Función de activación: Relu + optimizador: Adam

```
198/198 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - loss: 9.2060 - mse: 9.2060 - val_loss: 7.8849 - val_mse: 7.8849
Epoch 2/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 6.4743 - mse: 6.4743 - val_loss: 8.0304 - val_mse: 8.0304
Epoch 3/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - loss: 3.5494 - mse: 3.5494 - val_loss: 8.5321 - val_mse: 8.5321
Epoch 4/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 1.9604 - mse: 1.9604 - val_loss: 9.0051 - val_mse: 9.0051
Epoch 5/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 1.2072 - mse: 1.2072 - val_loss: 9.4065 - val_mse: 9.4065
Epoch 6/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 1.0196 - mse: 1.0196 - val_loss: 9.3657 - val_mse: 9.3657
Epoch 7/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.5976 - mse: 0.5976 - val_loss: 9.1948 - val_mse: 9.1948
Epoch 8/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.5557 - mse: 0.5557 - val_loss: 9.1024 - val_mse: 9.1024
Epoch 9/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 0.5369 - mse: 0.5369 - val_loss: 9.1058 - val_mse: 9.1058
Epoch 10/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - loss: 0.3374 - mse: 0.3374 - val_loss: 8.9960 - val_mse: 8.9960
<keras.src.callbacks.history.History at 0x7893f521d630>
```

Función de activación: Sigmoid + optimizador: Adam

```
198/198 ━━━━━━━━━━━━━━━━━━━━ 6s 18ms/step - loss: 14.0945 - mse: 14.0945 - val_loss: 8.4572 - val_mse: 8.4572
Epoch 2/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - loss: 9.1995 - mse: 9.1995 - val_loss: 8.3897 - val_mse: 8.3897
Epoch 3/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 6.6140 - mse: 6.6140 - val_loss: 8.4955 - val_mse: 8.4955
Epoch 4/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 6.1138 - mse: 6.1138 - val_loss: 9.1978 - val_mse: 9.1978
Epoch 5/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 4.3764 - mse: 4.3764 - val_loss: 9.4148 - val_mse: 9.4148
Epoch 6/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 3.0578 - mse: 3.0578 - val_loss: 10.0956 - val_mse: 10.0956
Epoch 7/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 2.6422 - mse: 2.6422 - val_loss: 9.7289 - val_mse: 9.7289
Epoch 8/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 3.1282 - mse: 3.1282 - val_loss: 10.1125 - val_mse: 10.1125
Epoch 9/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 2.1064 - mse: 2.1064 - val_loss: 10.3177 - val_mse: 10.3177
Epoch 10/10
198/198 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 1.4823 - mse: 1.4823 - val_loss: 10.1115 - val_mse: 10.1115
<keras.src.callbacks.history.History at 0x7894430dcca0>
```

RANDOM FOREST

Preprocesamiento

Utilizamos el mismo tipo de preprocesamiento que para el XGBoost.

Modelo

Utilizamos la versión para regresión "*RandomForestRegressor*".

Realizamos una búsqueda de hiperparametros mediante RandomizedSearchCV y con 10 folds.

Este modelo no tenía un coste computacional tan alto como el XGBoost, entonces decidimos que 10 folds era un número adecuado en cuanto a recursos que se usarán y el beneficio que produce.

Usamos dos hiperparametros fijos

3. criterion = squared_error: Usamos esta opción, el error cuadrático medio (MSE).
4. eval_metric= rmse : Usamos este parámetro fijo debido a que rmse, era la métrica que queríamos mejorar debido a que era la métrica que se tenía en cuenta en la competencia de kaggle.

Búsqueda de hiperparametros:

- 'max_depth': list(range(3, 15))
- 'learning_rate': [0.01, 0.1, 0.2]
- 'subsample': [0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
- 'n_estimators' : [50,100,150,200]

RMSE KAGGLE: 2.768

CASCADING - Árbol de regresión + XGBoost + LGBMRegressor

Preprocesamiento

Aplicamos el preprocesamiento de convertir la columna “project” a una columna numérica de modo tal que el modelo de aprendizaje pueda procesar los datos.

También lo que hicimos es crear una función que tokeniza y lemmetiza un texto pasado por parámetro y devuelve del mismo las 6 palabras más comunes que aparecían en él como una lista. Esto lo logramos con la librería de Spacy.

La función previamente mencionada la aplicamos a la columna “title”.

A la columna “description” le aplicamos una función que devolvia las oraciones que contenía un texto con sus palabras tokenizadas y lematizadas.

Esta vez poseemos un index para las oraciones que contiene todas las que se encontraron en las descripciones y su similitud con las otras oraciones del index fue menor a un 75%.

Y otro que representa una bolsa de palabras para los textos que aparecen en el título. Este está generado por todas las palabras diferentes y más usadas por cada título.

Luego representamos en columnas los identificadores de las oraciones y palabras en los index y en sus filas la frecuencia con la cual aparecen en estos textos.

Finalmente, le retiramos las columnas que ya no necesitamos.

Modelo

Usamos un modelo de cascading ya que posee un entrenamiento secuencial de modelos en el que cada modelo se entrena poniendo como características adicionales las predicciones de los modelos anteriores. Pensamos que esto haría que el ensamble pudiera captar relaciones más complejas entre los datos.

Utilizamos en primera instancia un árbol de regresión, debido a que al ser un modelo simple, podría captar las relaciones más simples entre las variables.

Seguido, entrenamos un XGBoost debido a que al ser un modelo más complejo, este podría captar mejor las relaciones entre los datos, refinar mejor las predicciones y corregir los errores del primer modelo.

Por último, utilizamos un LightGBM regressor para hacer las predicciones teniendo en cuenta el set de test.

Entrenamiento de los modelos

A todos los modelos se les aplicó una búsqueda de hiperparametros.

Búsqueda de hiperparametros DecisionTreeRegressor:

```
parametros = {'criterion':['squared_error'],

              'min_samples_split': list(range(4,21)),

              'min_samples_leaf': list(range(2,20)),

              'ccp_alpha':np.linspace(0,0.05,15),

              'max_depth':list(range(1,21)),

              'max_features': [0.5, 0.6, 0.7, 0.8, 0.9]}
```

Búsqueda de hiperparametros XGBRegressor:

```
parametros = {'objective': ['reg:squarederror'],

              'eval_metric': ['rmse'],

              'max_depth': list(range(3, 15)),

              'learning_rate': [0.01, 0.1, 0.2],

              'subsample':  [0.5, 0.6, 0.7, 0.8, 0.9, 1.0],

              'n_estimators' : [50,100,150,200],}
```

Búsqueda de hiperparametros LightGBM regressor

```
parametros = {'boosting_type': ['gbdt', 'dart', 'goss'],

              'num_leaves': [31, 50, 100],

              'learning_rate': [0.01, 0.1, 0.2],

              'min_data_in_leaf': randint(20, 100),

              'feature_fraction': [0.4, 0.5, 0.6, 0.7, 0.8],
```

```
            'max_depth':list(range(3, 15)),

            'lambda_l1': uniform(0.0, 1.0),

            'lambda_l2': uniform(0.0, 1.0),

            'n_estimators':  [100,150,200,250]}
```

RMSE KAGGLE: 2.944

Comparativa en cuanto a resultados con el set de entrenamiento

El RMSE con los datos del set de entrenamiento dio un valor de 2.097.

Esto puede significar que el modelo generó un overfitting respecto a los datos y no se adapta muy bien a los nuevos.

Integrante	Principales tareas realizadas	Promedio Semanal (hs)
Camila Lucia Avalos	XGBoost, ensamble, Random forest	4
Tomas Benitez Potochek	Naive Bayes, XGBoost, Red Neuronal	2
Cristian Roldan	XGBoost, Red Neuronal, Embeddings, Preprocesamiento	4
Roberta Sprenger	XGBoost, Red Neuronal	2