.UBAfiuba
FACULTAD DE INGENIERÍA

TA047 - Ciencia de Datos Dr. Ing. Rodríguez - 2°C 2024

Trabajo Práctico 2 : Story Points - Grupo 05 (DataPinto)

Integrantes

- Mejia Alan Roberto - 91161
- Prieto Pablo Alejandro - 91561
- Sosa Zoraida Flores - 87039

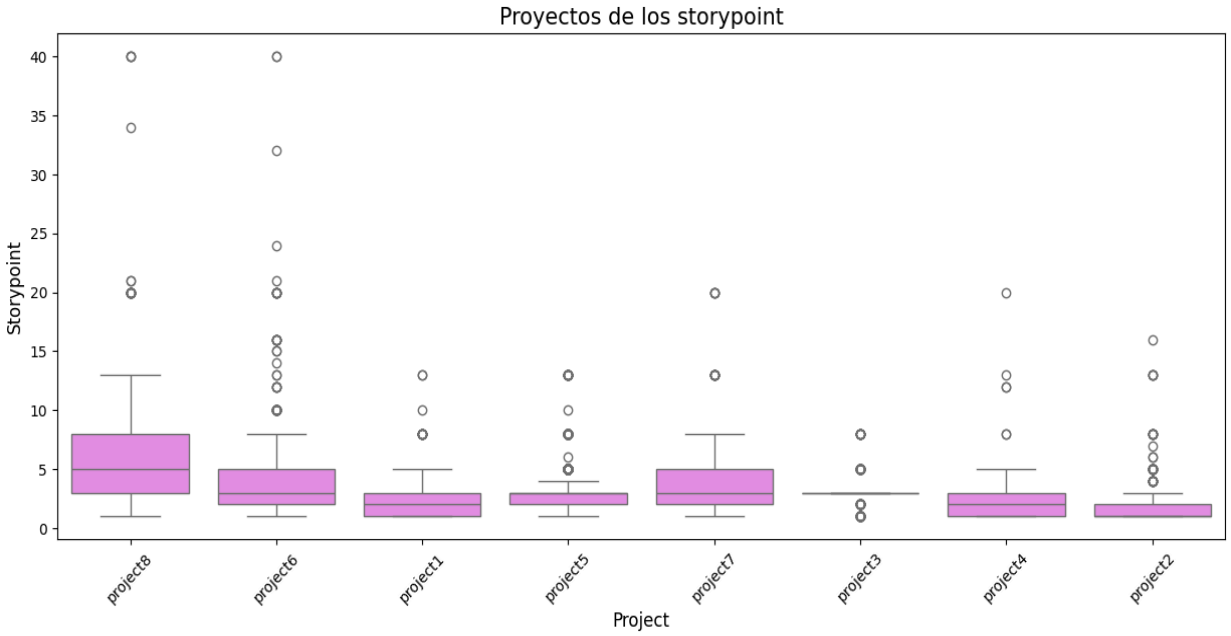
Introducción

En este trabajo práctico se va a utilizar un conjunto de datos que contiene una serie de casos de uso (user stories) de distintos proyectos y el número de story points que tiene asignado cada uno. Los story points indican la complejidad de cada tarea. El objetivo será predecir el story point de cada user story dado el texto que lo representa.

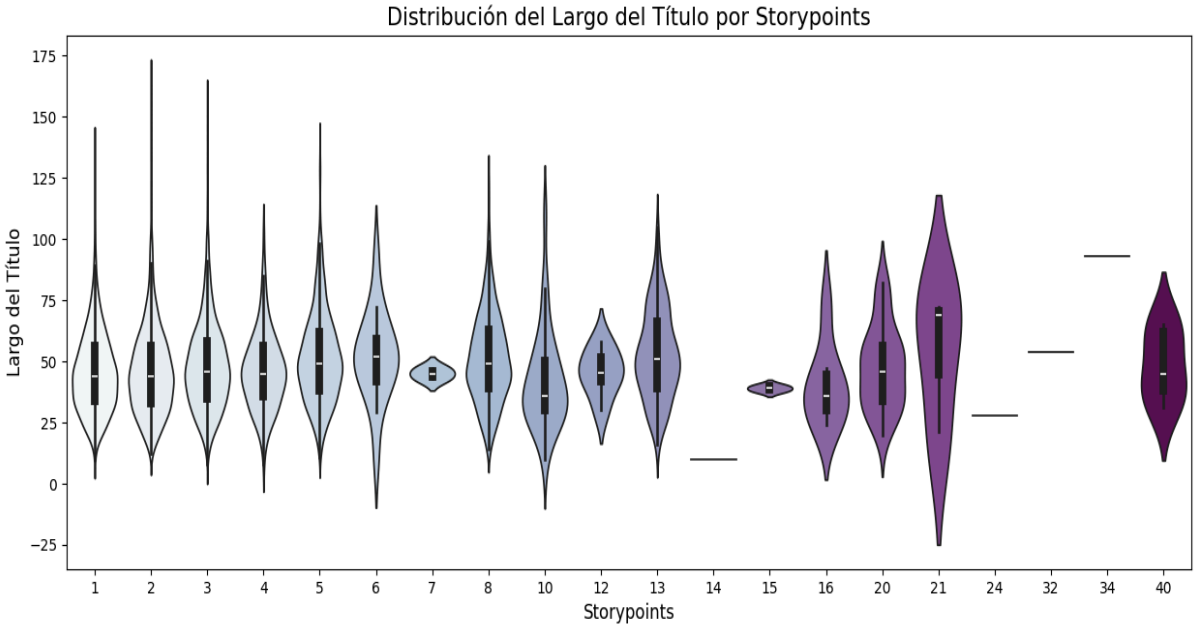
Análisis exploratorio

Se verificó que el dataframe no tenía registros nulos y tampoco filas duplicadas. Posteriormente se trabajó en las columnas title y description usando técnicas de NPL como stopwords, tokenizado

Verificamos que los proyectos con un promedio medio de 5 puntos (story points), corresponde al project 8



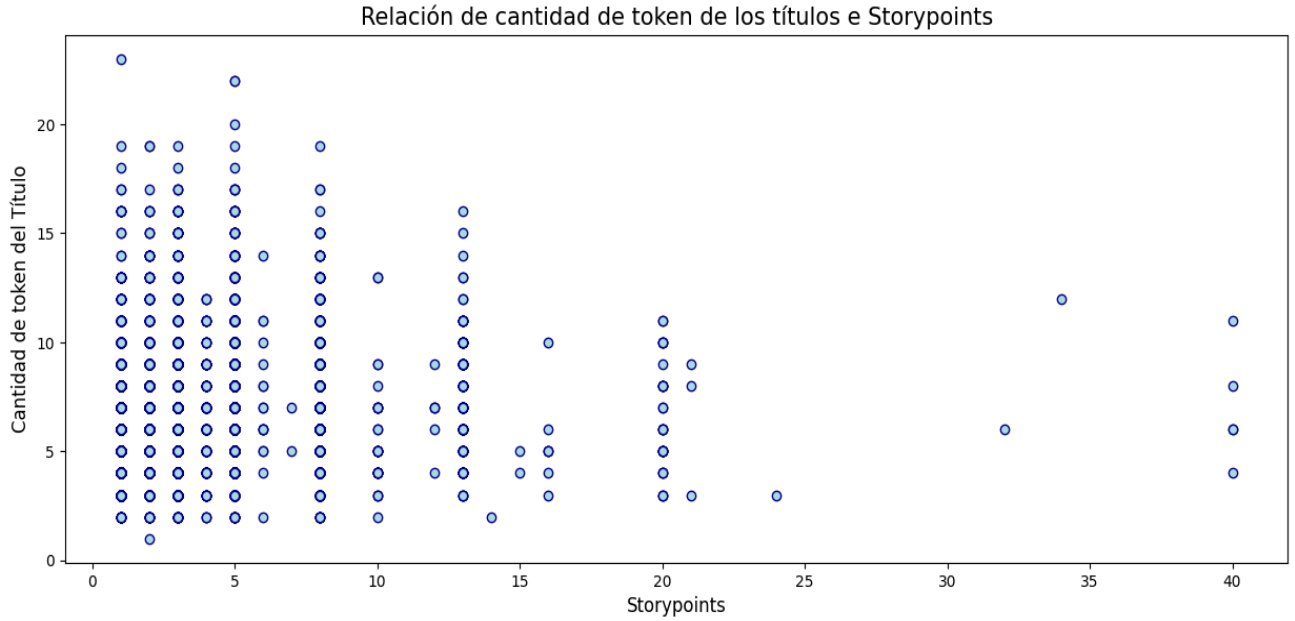
Luego se crearon las columnas title_largo y description_largo con el fin de ver la relación con storypoint, donde vemos que el largo del título con mayor densidad tiene un puntaje de storypoints=21 y su promedio es el más elevado a diferencia del resto.



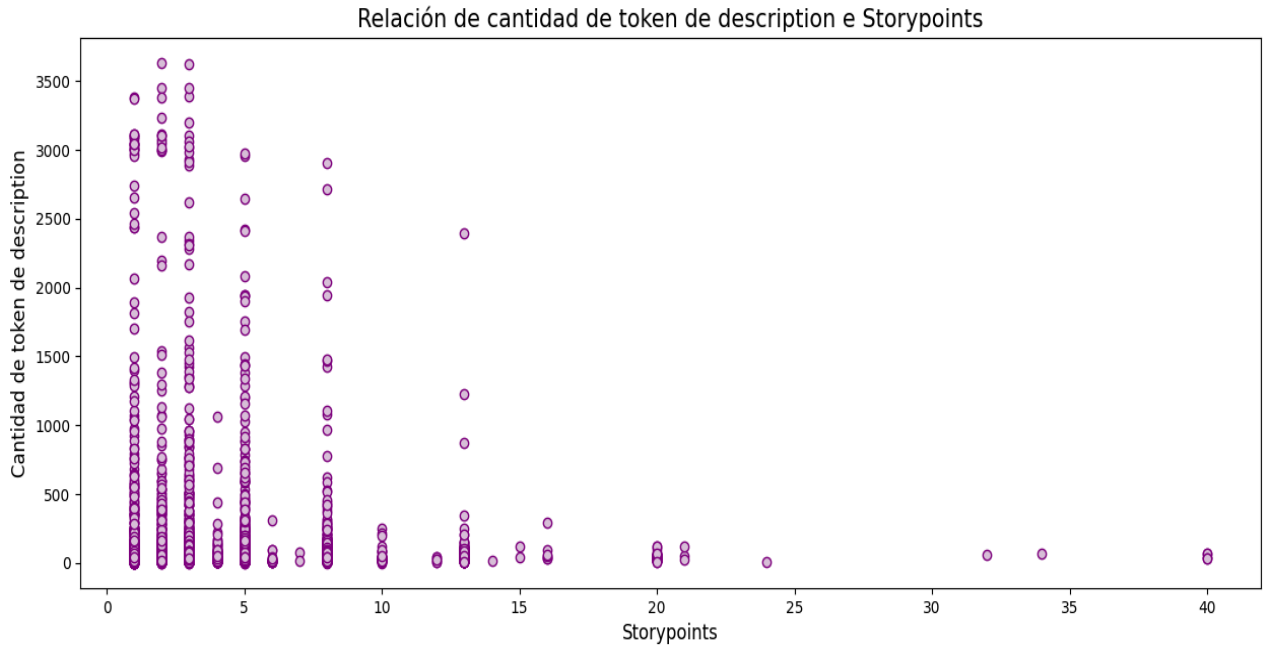
En cuanto a la longitud de las descripciones vemos que existe mayor densidad en aquellas longitudes donde storypoints es 13 y 16

A su vez se crearon las columnas title_token_cant y description_token_cant con el fín de ver la relación que tienen estas columnas respecto a storypoint

La relación del título y el puntaje, se observa que existe mayor cantidad de token en aquellos proyectos con menor complejidad



Sucede lo mismo en la relación de la descripción con el puntaje



Features

Se crearon los sgtes features con el fin de trabajar en un modelo más robusto

- title_largo
- description_largo
- title_token_cant
- description_token_cant

Procesamiento de datos

División de datos

Se realizó un split 80% train y 20% validation (y también 70% / 30%) con el fin de medir la performance de los distintos modelos con los que se va entrenar y a su vez procuramos en sacar los storypoint de validation que no están en train, ya que no puedo predecir valores que no se haya entrenado

Encodeo

Se realizaron one hot encoding para la columna project

Modelos

Se construyeron 5 modelos, para los cuales utilizamos la técnica de Cross Validation (KFolds, utilizando distintos valores) para buscar los mejores hiper parámetros, utilizando Random Search CV (para ciertas combinaciones al azar) o Grid Search (para probar todos los hiperparametros elegidos).

Bayes Naïve

Utilizando Cross Validation con Grid Search y TfidfVectorizer.

Mejores hiperparámetros: {'classifier_alpha': 5.0, 'vectorizermax_features': 20000, 'vectorizermin_df': 3, 'vectorizer_ngram_range': (1, 2)}

RMSE en validación: 2.663275797712949 y en kaggle: 3.03537

Random Forest

Para este modelo se trataron los texto de la sgte manera:

- Se pasó a minúscula
- Se eliminó signos de puntuación
- Se eliminó las palabras vacías usando stopwords

Se agregaron los features, mencionados anteriormente y se ha dividido los datos como lo mencionado anteriormente

Se probarón los sgtes modelos:

1. TF-IDF con optimización de búsqueda de hiperparametros con Random-Search
2. TF-IDF ajustando los hiper parámetros del modelo anterior con Random-Search
3. TF-IDF con optimización de búsqueda de hiperparametros con Grid-Search y
4. Count-Vectorized con optimización de búsqueda de hiperparametros con Random-Search

En el cual probando estos modelos el que mejor score nos dió fue la opción 4, donde:

Mejores hiperparametros encontrados fueron:

- rf__n_estimators: 40
- rf__min_samples_split: 10
- rf__min_samples_leaf: 2
- rf__max_features: sqrt
- rf__max_depth: None

Donde el modelo tiene una mejor puntuación de error cuadrático medio negativo (negativo de MSE): -7.513438300668642 y un rmse = np.sqrt(7.51) = 2.74

En kaggle: 2.80281

XGBoost

Para este modelo se trataron los texto de la sgte manera:

- Se pasó a minúscula
- Se eliminó signos de puntuación
- Se eliminó las palabras vacías usando stopwords

En este caso quisimos probar en trabajar solo con la columna description

Se probaron los sgtes modelos:

1. TF-IDF hiperparametros por defecto
2. TF-IDF con 100 árboles
3. TF-IDF con 100 árboles y aumentando su profundidad a 4
4. TF-IDF optimizado con cross-validation

De los 4 modelos anteriores, el que mejor puntuación nos dió fue el de la opción (4) con los sgtes hiperparametros:

- max_depth: 15
- alpha: 10
- n_estimators: 300
- learning_rate: 0.1
- colsample_bytree: 0.3

Donde nos dió un rmse en validation de: 2.528934 y en kaggle: 2.81617

Red Neuronal con Tensor Flow y Keras.

Probamos 2 tipos de redes con TensorFlow y Keras utilizando distintas capas y optimizador de la red.

Primero armamos una Red Neuronal con 3 capas secuenciales y el optimizador Adam (uno de los estándares que es bastante bueno). Comenzando con la primera capa de densidad 128 y activación “relu” (la más común para regresión) y la segunda parametrizada para buscar el mejor en el Cross Validation (también activación relu), y la última capa de densidad 1 y activación linear.
Combinándolo con un Pipeline con TfidfVectorizer (con un preprocessor que llama a una función *lemmatizationAndStopwords* definida por nosotros para lematizar las palabras y procesar los stopwords) y TruncatedSVD.

Mejores hipérparametros encontrados fuerón:
{'tfidf__use_idf': False, 'tfidf__sublinear_tf': False, 'tfidf__stop_words': ['english'], 'tfidf__ngram_range': (1, 2), 'tfidf__min_df': 2, 'tfidf__max_features': 1000, 'tfidf__max_df': 0.8, 'tfidf__analyzer': 'word', 'svd__random_state': 42, 'svd__n_components': 100, 'keras_regressor__learning_rate': 0.001, 'keras_regressor__epochs': 10, 'keras_regressor__dropout_rate': 0.4, ‘keras_regressor__dense_units': 32, 'keras_regressor__batch_size': 16}

RMSE en validación: 2.8201966753545467 y en kaggle: 2.93019

Segundo armamos una Red Neuronal más compleja con 4 capas secuenciales y el optimizador parametrizado (con las opciones: Adam, Adamax, RMSprop, Nadam, SGD).
Comenzando con la primera capa de densidad 128 y activación “relu” (la más común para regresión), luego BatchNormalization (para normalizar luego de cada

capa y estabilizar) y un Dropout de 30% (apaga 30% aleatoriamente las unidades) para reducir el sobreajuste.
La segunda capa con densidad 64 y activación elu. Regularizando los pesos con L2 (penalizando pesos grandes para evitar el sobreajuste), seguido BatchNormalization y Dropout 40%.
La tercera capa de densidad 32 y activación relu.
La última capa de densidad 1 y activación linear.
Aplicando en el modelo de KerasRegressor la regulación Early Stopping, para monitorear las pérdidas y restaurando los mejores pesos encontrados si no mejora en 5 épocas (deteniendo el entrenamiento).
Combinándolo con un Pipeline con TfidfVectorizer (con un preprocessor que llama a una función *lemmatizationAndStopwords* definida por nosotros para lematizar las palabras y procesar los stopwords) y TruncatedSVD (luego de la reducción de dimensiones aplicamos StandardScaler para estandarizar los features para la estabilidad y eficiencia del entrenamiento).

En este caso, el mejor optimizador fue el SGD, con 10 épocas y 256 batch size.

Mejores hipérparametros encontrados fuerón:
{'tfidf__use_idf': True, 'tfidf__sublinear_tf': True, 'tfidf__stop_words': ['english'], 'tfidf__ngram_range': (1, 2), 'tfidf__min_df': 3, 'tfidf__max_features': 3000, 'tfidf__max_df': 0.75, 'tfidf__analyzer': 'word', 'svd__random_state': 42, 'svd__n_components': 100, 'keras_regressor__optimizer_name': 'SGD', 'keras_regressor__learning_rate': 0.01, 'keras_regressor__epochs': 10, 'keras_regressor__batch_size': 256}

RMSE en validación: 3.0674813522442883 y en kaggle: 2.91698

Ensamble de al menos 3 modelos elegidos por el grupo.

Probamos 2 tipos de ensamble usando Stacking.

Primero armamos un Stacking con los estimadores XGBRegressor, RandomForestRegressor y ElasticNet, con GradientBoostingRegressor como metamodelo. Combinándolo con un Pipeline con TfidfVectorizer (con un preprocessor que llama a una función *lemmatizationAndStopwords* definida por nosotros para lematizar las palabras y procesar los stopwords) y TruncatedSVD.

Mejores hipérparametros encontrados fuerón:
{'tfidf__use_idf': False,
'tfidf__sublinear_tf': False,

'tfidf__stop_words': ['english'],
'tfidf__ngram_range': (1, 2),
'tfidf__min_df': 2,
'tfidf__max_features': 2000,
'tfidf__max_df': 0.8,
'tfidf__analyzer': 'word',
'svd__random_state': 42,
'svd__n_components': 100,
'stacking__xgb_regressor__random_state': 42,
'stacking__xgb_regressor__objective': 'reg:squarederror',
'stacking__xgb_regressor__n_estimators': 100,
'stacking__xgb_regressor__max_depth': 3,
'stacking__xgb_regressor__learning_rate': 0.01,
'stacking__rnd_regressor__random_state': 42,
'stacking__rnd_regressor__n_jobs': -1,
'stacking__rnd_regressor__n_estimators': 300,
'stacking__rnd_regressor__min_samples_split': 2,
'stacking__rnd_regressor__min_samples_leaf': 1,
'stacking__rnd_regressor__max_features': 'sqrt',
'stacking__rnd_regressor__max_depth': 7,
'stacking__rnd_regressor__criterion': 'poisson',
'stacking__rnd_regressor__bootstrap': True,
'stacking__final_estimator__random_state': 42,
'stacking__final_estimator__n_estimators': 200,
'stacking__final_estimator__max_depth': 4,
'stacking__elasticnet__random_state': 42,
'stacking__elasticnet__l1_ratio': 0.1,
'stacking__elasticnet__alpha': 1.0}

RMSE en validación: 2.9470424378844795 y en kaggle: 2.97331

Segundo armamos un Stacking con los estimadores XGBRegressor, RandomForestRegressor y LGBMRegressor, con ElasticNet como metamodelo. Combinándolo con un Pipeline con TfidfVectorizer (con un preprocessor que llama a una función *lemmatizationAndStopwords* definida por nosotros para lematizar las palabras y procesar los stopwords) y TruncatedSVD.

Mejores hipérparametros encontrados fuerón:
{'tfidf__use_idf': True,
'tfidf__sublinear_tf': True,
'tfidf__stop_words': ['english'],
'tfidf__ngram_range': (1, 2),
'tfidf__min_df': 2,

'tfidf__max_features': 2000,
'tfidf__max_df': 0.75,
'tfidf__analyzer': 'word',
'svd__random_state': 42,
'svd__n_components': 400,
'stacking__xgb_regressor__random_state': 42,
'stacking__xgb_regressor__objective': 'reg:squarederror',
'stacking__xgb_regressor__n_estimators': 300,
'stacking__xgb_regressor__max_depth': 3,
'stacking__xgb_regressor__learning_rate': 0.01,
'stacking__rnd_regressor__random_state': 42,
'stacking__rnd_regressor__n_jobs': -1,
'stacking__rnd_regressor__n_estimators': 300,
'stacking__rnd_regressor__min_samples_split': 2,
'stacking__rnd_regressor__min_samples_leaf': 1,
'stacking__rnd_regressor__max_features': 'sqrt',
'stacking__rnd_regressor__max_depth': 7,
'stacking__rnd_regressor__criterion': 'squared_error',
'stacking__rnd_regressor__bootstrap': True,
'stacking__lgbm_regressor__subsample': 0.8,
'stacking__lgbm_regressor__random_state': 42,
'stacking__lgbm_regressor__num_leaves': 50,
'stacking__lgbm_regressor__n_estimators': 100,
'stacking__lgbm_regressor__min_child_samples': 5,
'stacking__lgbm_regressor__max_depth': 3,
'stacking__lgbm_regressor__learning_rate': 0.2,
'stacking__lgbm_regressor__colsample_bytree': 0.7,
'stacking__final_estimator__random_state': 42,
'stacking__final_estimator__l1_ratio': 0.5,
'stacking__final_estimator__alpha': 10.0}

RMSE en validación: 3.0575858141040166 y en kaggle: 3.12976

Conclusión

	Random-F	XGBoost	Bayes-N	Red N.	Ensamble
RMSE	2.74	2.5289	2.6632	3.0675	2.9470
Kaggle	2.8028	2.8161	3.303537	2.91698	2.97331

De todos los modelos que probamos el que mejor puntaje nos dió dentro de la competencia kaggle es el de Random Forest count vectorized con optimizado con Random-Search.

Durante los entrenamientos tuvimos que ajustar los hiper parámetros mediante el uso de la técnica Random Search, ya que el tiempo de entrenamiento de Grid-Search era muy extenso y no llegamos a obtener resultados (por el consumo de recursos que requería).

También nos ajustamos con la cantidad de iteraciones y los folds, para lograr resultados con menor tiempo (aún así probamos valores kfolds entre 3 y 20, para probar la mayor cantidad de combinaciones posibles).