Predicción de la productividad de una empresa de producción de ropa Q1 2022-23

Integrantes:
Daniel Muñoz Arroyo
Cristian Sánchez Estapé

Índice

1. Descripción del problema y sus datos

China es el principal exportador de ropa del mundo, con una de las mayores industrias en cuanto a dimensiones y calidad se refiere. Este sector mueve mucho dinero, en concreto, unos 577 mil millones de dólares en 2022, según un informe de *The Business Research Company* [5] y, por lo tanto, es natural que muchas empresas del sector quieran participar en ello lo máximo posible. Sin embargo, la industria textil depende de tareas no mecanizadas, con lo cual la producción depende mucho del rendimiento de los trabajadores. Es por ello que merece la pena llevar un registro de los datos asociados a los equipos, departamentos, salario e incentivos que afectan (e influyen) a dichos trabajadores. Todo esto es lo que motiva la existencia de este dataset, que recoge ese tipo de datos (entre enero y marzo del 2015) y con los cuales trataremos de predecir la productividad de los distintos equipos involucrados en dicha actividad. Nuestra tarea, por lo tanto, será de regresión. En cuanto al conjunto de datos, la información recogida es la siguiente:

Atributo	**Descripción**
date	Fecha en formato MM-DD-YYYY
day	Día de la semana (lunes-domingo)
quarter	Indica que semana del mes es con quarterX donde X es un valor entero de 1 a 5.
department	Departamento asociado a la muestra
team	Equipo asociado a la muestra
targeted_productivity	Productividad esperada, según autoridad competente, por equipo y día
smv	Tiempo esperado para realizar cada tarea (*Standard Minute Value*)
wip	Número de ítems sin acabar por cada producto (*Work In Progress*)
over_time	Representa el tiempo extra (en minutos) consumido por cada equipo
incentive	Incentivo financiero (en takas[1]) para motivar un equipo a hacer una tarea
idle_time	Cantidad de tiempo de interrupción de la producción por razones externas
idle_men	Cantidad de trabajadores afectados por la interrupción de la producción
no_of_style_changes	Número de cambios en el estilo de un determinado producto
no_of_workers	Número de trabajadores en un equipo determinado
actual_productivity	El porcentaje (dentro del rango 0-1) de productividad del equipo de la muestra. Corresponde a nuestra variable objetivo

Figura 1: tabla con los atributos del dataset y su correspondiente descripción

Visto esto, pasemos brevemente a mencionar los estudios previos (e información asociada) que hemos encontrado acerca del presente problema.

[1] Divisa de Bangladesh.

2. Estudios previos acerca del problema

Existen varios estudios para este problema. Uno de ellos [1] hace clasificación. Para ello, preprocesa los datos (imputación de *missing values*, binarización de variables categóricas, creación de nuevas variables y balanceo de los datos) y visualiza las variables tratando de mostrar sus distribuciones y outliers. Después, clasifica la variable objetivo (*actual_productivity*) en 3 clases fruto de calcular la diferencia entre la variable objetivo y la productividad deseada (*targeted_productivity*), obteniendo así valores igual a 0 si la deseada es la misma que la obtenida, superior a 0 si la obtenida es mayor que la deseada o inferior a 0 si la obtenida no llega a la esperada. Posteriormente a un ligero preprocesamiento de las clases de la variable objetivo, hace la partición de *test* y *train*, realiza un escalado y utiliza modelos tales como la regresión logística, random forest o knn para resolver el problema. Una vez entrenados, el mejor modelo que obtiene es el *random forest* con un R^2 de 0.998571 para el conjunto *train* y de 0.854286 para el conjunto de *test*, lo que en definitiva implica unos buenos resultados.

De este estudio podemos ver que los modelos vistos tienen unos buenos resultados, así que en el nuestro nos gustaría mirar si pueden alcanzar tal calidad, a la vez que se centra más en concretar exactamente qué valores puede tomar *actual_productivity*. Adicionalmente, también tomamos nota del tipo de preprocesamiento que se efectúa. Finalmente, también queremos comprobar cómo de susceptible es el problema de ser atacado mediante la regresión.

Por otro lado, el estudio asociado al dataset [2] presenta algunas consideraciones interesantes: en primer lugar, su análisis del problema se realiza sobre el mismo conjunto de datos, pero con menos variables (faltan *day* y *quarter*); su preprocesamiento es relativamente distinto al anterior (hay creación de nuevas variables y escalado solo en algunas de ellas) y, finalmente, enfocando el problema como uno de regresión lineal, aplican un perceptrón multicapa para tratar de predecir la variable objetivo. Si bien del anterior estudio no hemos recogido más información que aquella concerniente al resultado de sus modelos y su enfoque para el preprocesamiento, de este tomamos nota del preprocesamiento ejecutado y del modelo aplicado (que es único y central para el estudio). Referente a esto último, se aplica un perceptrón de 2 capas ocultas con una serie de tamaños y funciones de activación relacionados con las características del problema (número de variables, tipo de preprocesamiento efectuado, etc.), de los cuales tomamos nota y sobre los cuales iteraremos: trataremos de cambiar tamaños, funciones de activación e hiperparámetros asociados al modelo usado.

Finalmente, en cuanto al error se refiere, nosotros utilizaremos el error R^2 para juzgar adecuadamente nuestros resultados

3. Exploración de los datos

Carga de datos y visualización preliminar

Una vez cargados los datos descargados de UCI, hacemos una visualización preliminar de algunas de sus primeras instancias y de la estadística descriptiva del conjunto, con tal de ver el formato y características de las variables. Después, hicimos la partición de datos de *train* y *test* con una proporción de 70-30 con tal de garantizar que el conjunto de entrenamiento tenga un tamaño suficiente como para entrenar los modelos adecuadamente, a la vez que separamos la variable objetivo del resto. Hecho esto, procedemos a visualizar los datos.

Visualización

Para la visualización de las variables, hemos utilizado *profile_reports* de *pandas*, ya que muestra mucha información básica de las variables. Lo primero en lo que nos fijamos es en la corrección de las variables. A simple vista, podemos encontrar anomalías o valores extraños en las siguientes:

- *department*: se puede observar que aparecen dos valores de "*finishing*" como si fueran opciones diferentes.
- *no_of_workers*: al ser un número de trabajadores, debería de ser un valor entero, ya que no puede haber "media persona". En cambio, en la visualización de esta variable[2], se puede constatar valores como 56.5.
- *actual_productivity*: en el dataset se indica que los valores van en el rango [0,1], pero si se mira su histograma, podemos ver que tiene valores por encima de 1. Esto puede ser debido a que en algunos casos la productividad es mayor que la deseada. Aun así, el máximo debería ser 1.

En otro orden de cosas, con la matriz de valores perdidos también se puede ver que, de todas las variables, solo tenemos valores extraviados en *wip* para el 42% de las instancias.
Adicionalmente, otra cosa a tener en cuenta es que, de todas las variables que hay, *date* sería parcialmente redundante con las variables actuales (*day* y *quarter* principalmente), y pasaría a serlo totalmente si se tuviera una tercera variable para el mes.
Finalmente, también consideramos que, para las variables numéricas, habrá que comprobar la presencia de outliers, habrá que aplicarles un escalado y, para aquellas que no sean discretas, habría que mirar de tratar la curtosis y las simetrías (muchas de las variables no siguen distribuciones normales y aparecen desviadas).

Preprocesamiento

A continuación, explicitamos cada uno de los pasos llevados a cabo.

[2] Se puede comprobar accediendo a *more details → common values*

Corrección de valores

Basándonos en la visualización, tratamos las variables de la siguiente forma:

- En *department* concentramos todos los valores en "*finishing*".
- En *no_of_workers* redondeamos sus valores al alza.
- En *actual_productivity* hemos acotado todos los valores al rango [0,1].

Esto garantizaría que las variables tengan valores correctos de acuerdo a su definición.

Tratamiento de valores perdidos

Dado que *wip* es la única variable con valores perdidos, dado que desconocemos el origen de la desaparición de dichos valores y teniendo en mente que la naturaleza de la variable es discreta (véase su descripción en la figura 1), procedemos a imputarlos con su mediana, lo cual, a nuestro parecer, es la acción más neutra y mantendría la consistencia de los datos.

Eliminación de variables irrelevantes/Creación de nuevas variables

Dado lo observado, crearíamos la variable *month* que, junto con *day* y *quarter*, expresarían la misma información que *date*, con lo cual (y teniendo en cuenta que el año es siempre el mismo) procederíamos a borrarla.

Codificación de variables categóricas

Como las variables categóricas no pueden ser texto, las binarizamos. Para ello hemos utilizado *get_dummies* de *pandas*. Algo a tener en cuenta es que esta función puede eliminar la primera columna de cada variable para evitar combinaciones lineales. Nosotros no lo haremos, puesto que podríamos perder información, además del hecho que los modelos que usaremos no tienen problemas con este tipo de columnas.

Análisis y tratamiento de outliers

Para ver si hay outliers, visualizamos los *boxplots* de las variables numéricas (tanto discretas como continuas):

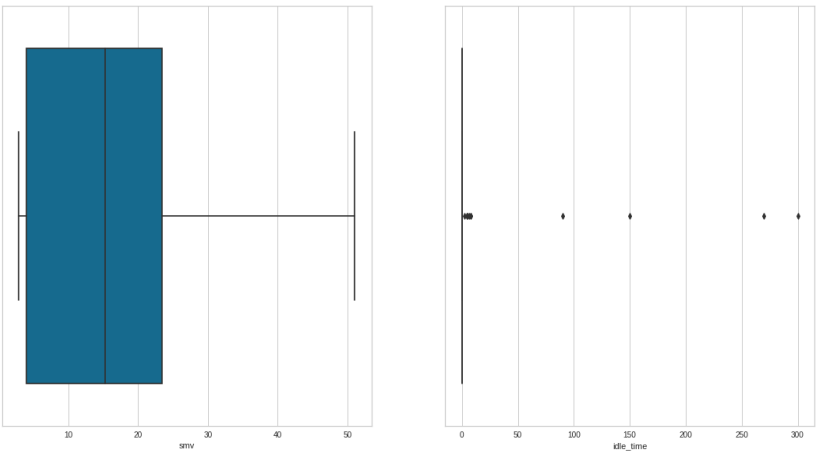

Figura 2: boxplots de smv e idle_time

Como se puede ver en la figura 3, hay variables con outliers (como en *idle_time*) y otras sin (como en *smv*). El proceso que hemos utilizado para tratarlos ha sido el mismo para todas: decidimos sustituirlos por un valor máximo o mínimo dependiendo de si pertenece al extremo superior o inferior (asumimos que los outliers no son valores erróneos, sino que pueden darse naturalmente para las variables y que serían igualmente extremos para los datos recogidos). Al aplicar estas modificaciones, obtenemos resultados tales como:

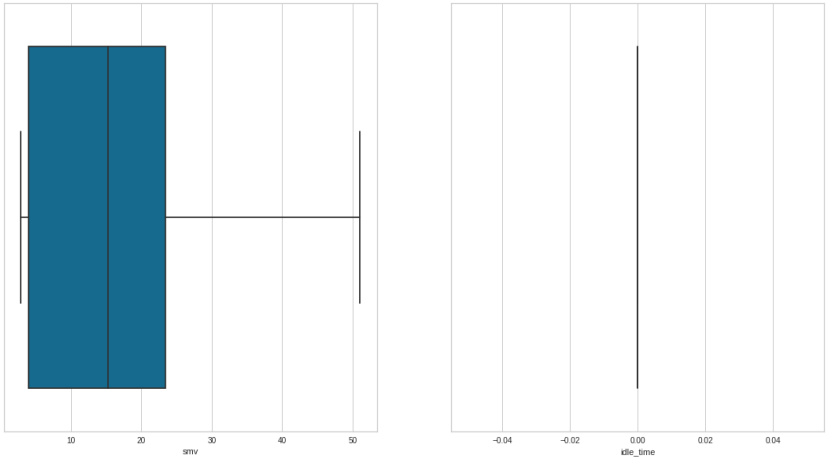

Figura 3: boxplots de smv e idle_time

Como se pueden observar en la figura 4, ya no aparecen outliers, con lo cual el tratamiento daría sus frutos. Sin embargo, lo que ocurriría es que *idle_time* e *idle_men* solo tomarían un único valor para todas las instancias (0), con lo cual la eliminamos.

Corrección de asimetrías y/o curtosis

Una vez tratados los outliers, realizamos una visualización previa del tipo de distribuciones que siguen las variables numéricas (excluyendo las discretas, esto es, *wip*, *no_of_workers* y *teams*) con tal de encontrar asimetrías y comprobar su curtosis. Ello se aplica con tal de suavizar (y, si fuera posible, normalizar) las distribuciones de las distintas variables, con tal de reducir la complejidad asociada a los datos. Una vez hecho esto, observamos figuras tales como la siguiente:

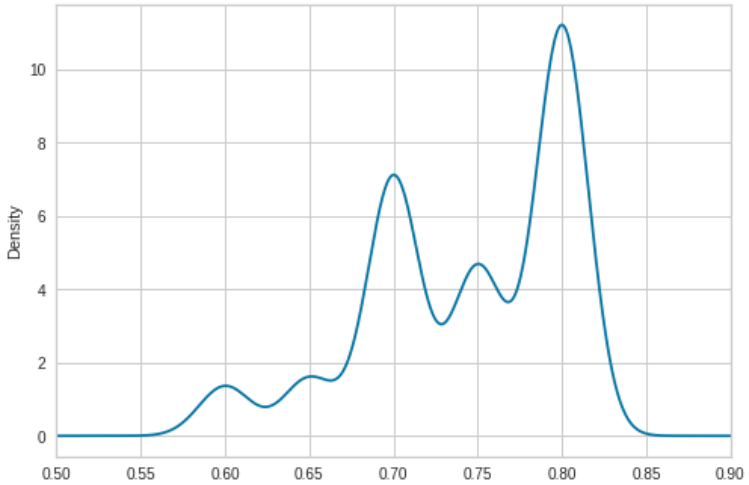

Figura 4: distribución de la variable targeted_productivity

Tal y como se observa, todas las variables (en mayor o menor medida) siguen distribuciones no normales. Además, si computamos su *skewness*, vemos reafirmada esta observación. Por ello, nos disponemos a aplicar el método de Yeo-Johnson con tal de tratar de "normalizar" la distribución que siguen. Posteriormente, si visualizamos de nuevo su *skewness*, vemos que esta disminuye, con lo que ciertamente hemos reducido su asimetría. Sin embargo, su curtosis se aleja del 0, lo que implica que las distribuciones no se acercan a una distribución normal clásica (o bien corresponden a distribuciones de tipo *platykurtic* o bien de tipo *leptokurtic* [3]).

Normalización de variables

La normalización de variables es importante para que todas estén en un rango de [0,1], logrando así que tengan todas la misma importancia de cara a los modelos o tomen valores aceptables para aplicarles el PCA o el TSNE. Para ello, aplicaremos un *MinMaxScaler* a todas las variables numéricas (tanto discretas como continuas).

Preprocesamiento del conjunto de test

Todo el preprocesamiento descrito se ha hecho para la partición de los datos de *train*, incluyendo la variable objetivo (para la que solo se efectúa la corrección de sus valores). Una vez tenemos los valores correspondientes y los *scalers* y métodos entrenados, replicamos el proceso para los datos de *test*.

PCA y TSNE

El análisis del PCA del presente conjunto de datos nos ayudará a conocer la potencial separabilidad (y, por lo tanto, la complejidad) del mismo. Para ello, procedemos a estudiarla para el conjunto de *train*. Lo primero que se puede destacar es que, con dos componentes, no somos capaces de alcanzar a explicar el 40% de los datos de entrenamiento, por lo que no podemos confiar demasiado en la información que podamos visualizar. Sin embargo, si representamos estas dos primeras componentes, observamos lo siguiente:

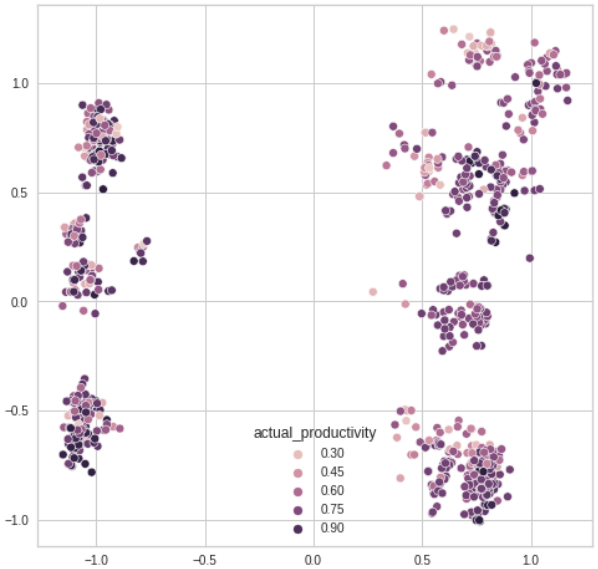

Figura 5: visualización del PCA para dos componentes

De acuerdo con esta figura, el análisis de PCA parece indicar que la complejidad del conjunto de datos es potencialmente elevada, dado que la separabilidad no parece factible bajo ninguna circunstancia (los datos aparecen demasiado entremezclados entre sí, formando pequeñas agrupaciones distanciadas entre ellas). Teniendo esto en mente, consideramos visualizar el TSNE del mismo conjunto, con tal de determinar si un análisis no lineal puede desvelar más información acerca del conjunto de datos tratado:

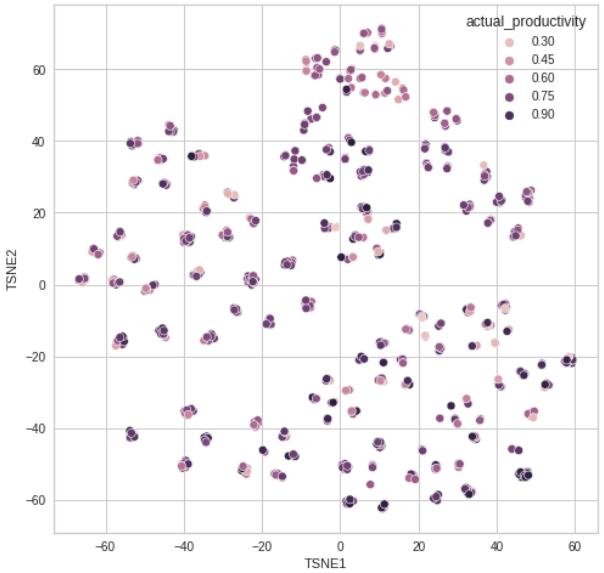

Figura 6: visualización del TSNE para dos componentes

Tal y como se observa, el TSNE también nos lleva a la creación de pequeños grupos heterogéneos de instancias de la variable objetivo. Por ello, ambos análisis parecen indicar que la complejidad del conjunto de datos es bastante elevada, por lo que es probable que tanto con modelos lineales como no lineales tengamos dificultades para realizar predicciones de calidad.

4. Protocolo de remuestreo

Para todos los modelos, el procedimiento que utilizaremos empezará buscando los mejores hiperparámetros para los datos de *train*. Los valores explorados para cada hiperparámetro tratan de incluir, dentro de sus márgenes, aquellos que den lugar al mejor modelo posible. Además, este entrenamiento se realiza una única vez, con lo cual no hay una exploración ni afinamiento exhaustivo de los hiperparámetros y sus valores, o lo que es lo mismo: la elección de los rangos de valores contempla que el entrenamiento solo se realice una vez.
Dicho esto, una vez entrenado el modelo, juzgamos con validación cruzada su desempeño sobre los datos de *train* y de *test* para ver cómo de bueno es, y ello lo haremos mediante la métrica de R^2. Además, dichas particiones se harán de forma que ambas tengan instancias seleccionadas aleatoriamente, lo que garantizará la adecuación de ambas particiones.
Con esto, estableceremos los siguientes valores para todos los modelos:

- **cv**: este valor corresponde al número de particiones que utilizaremos a la hora de hacer validación cruzada. Hemos escogido 5, ya que como hay 837 instancias de *train*, consideramos que tener 5 particiones con unas 167 instancias cada una es una cantidad adecuada para que la validación cruzada se efectúe adecuadamente.
- **niters**: este valor corresponde al número de iteraciones que se harán a la hora de buscar los mejores hiperparámetros. Emplearemos 10, puesto que consideramos que es un valor que, para el tamaño del conjunto de datos y la cantidad de variables con que se trabaja, nos asegura una buena solución.
- **maxiters**: Este valor corresponde al número máximo de iteraciones que se harán a la hora de buscar la mejor solución en algunos modelos. Emplearemos 2500, ya que consideramos que el conjunto de datos es de una dimensión tal que hacer más de estas iteraciones sería innecesario.

Una vez entrenado el modelo y vistos los errores de validación cruzada de *train* y de *test*, haremos una gráfica en la que miraremos los valores predichos con respecto a los valores reales. Finalmente, con el *permutation_importance* haremos una gráfica en la que se pueda ver qué atributos son los más relevantes para el modelo.

5. Modelos lineales

A continuación presentamos el entrenamiento de modelos de regresión lineal, de regresión ridge y de SVM con kernel cuadrático.

Regresión Lineal

Sabemos que es el modelo lineal más simple posible, además de no requerir de exploración hiperparamétrica alguna (los valores por defecto son necesarios si se quiere entrenar el modelo adecuadamente, puesto que corresponden a características del modelo, tal como el cálculo del *intercept* o la necesidad de copiar los datos de entrenamiento para no sobreescribir los originales, por ejemplo). Con esto, los resultados del entrenamiento son los siguientes:

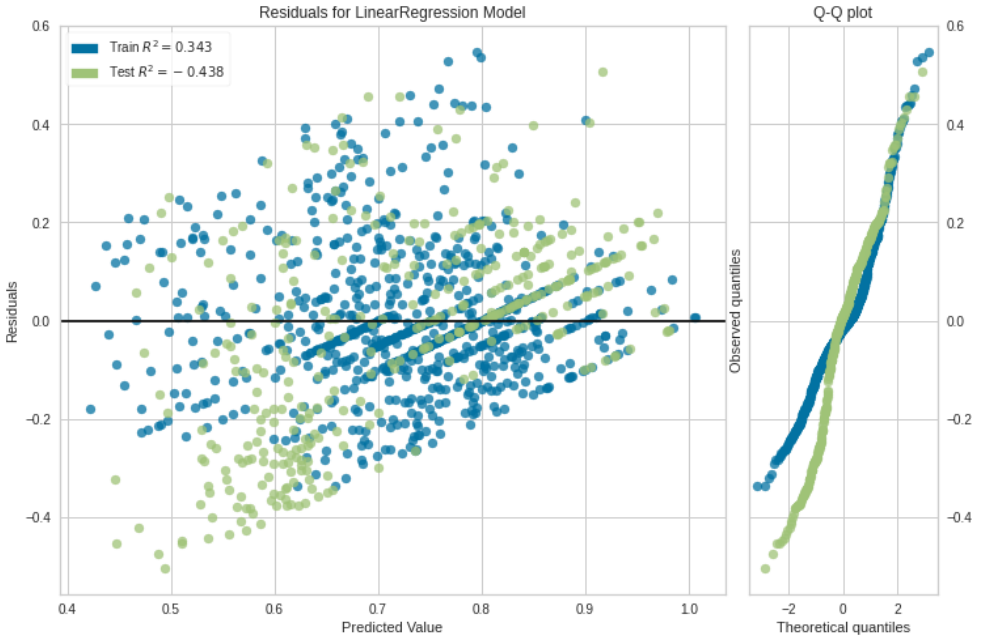

Figura 7: resultado del entrenamiento de un modelo de regresión lineal

Por un lado, el *residuals plot* da una muy mala noticia acerca de los datos: para el conjunto de *train*, los datos aparecen relativamente concentrados, por lo que el modelo tendría dificultades para realizar las predicciones adecuadamente; para el conjunto de *test*, vemos cómo, a pesar de encontrarse más distribuidos, los datos siguen estando concentrados en determinados núcleos, con lo cual el modelo no lo tiene nada fácil para realizar las predicciones debidamente.
Por otro lado, el *Q-Q plot* da noticias semejantes, puesto que vemos cómo el conjunto de test no llega ni a poder reproducir adecuadamente esa diagonalidad que en el conjunto de *train* se consigue replicar deficientemente. Ello implica que la distribución en el error de los datos no es gausiana, con lo cual la regresión lineal no es un modelo adecuado para el problema que aquí atacamos.
Finalmente, ello provoca que el error de test sea bastante malo, alcanzando solo un valor de 0.343, mientras que el error de test ni siquiera alcanza un valor positivo, llegando a -0.438. Por todo ello, concluimos que este modelo no es adecuado para resolver el problema.

Regresión Ridge

Para este modelo, de todos sus hiperparámetros, decidimos explorar solamente aquel relacionado con la regularización de los datos (*lambda*), y para el cual hemos escogido un rango de valores entre 0.0001 y 200. La mejor lambda, según el entrenamiento del modelo con validación cruzada, es de 1.

En cuanto a la selección del hiperparámetro de regularización se refiere, cabe destacar su importancia con relación al efecto que este tiene sobre los datos. La gráfica de *AlphaSelection* muestra el efecto que la selección de determinados valores de lambda tiene sobre la regularización del conjunto de datos. Finalmente, consideramos que una *lambda* de 1 tiene sentido según la regularización que este tendría sobre los datos.

En este sentido, el error de *train* obtenido después de hacer una validación cruzada es de 0.295, el cual es terrible e incluso peor que el de la regresión lineal. En cuanto al error de test, se obtiene un valor de -0.323, el cual, si bien es muy malo, es mejor que el de la regresión lineal. Dicho esto, veamos el resultado de las predicciones realizadas:

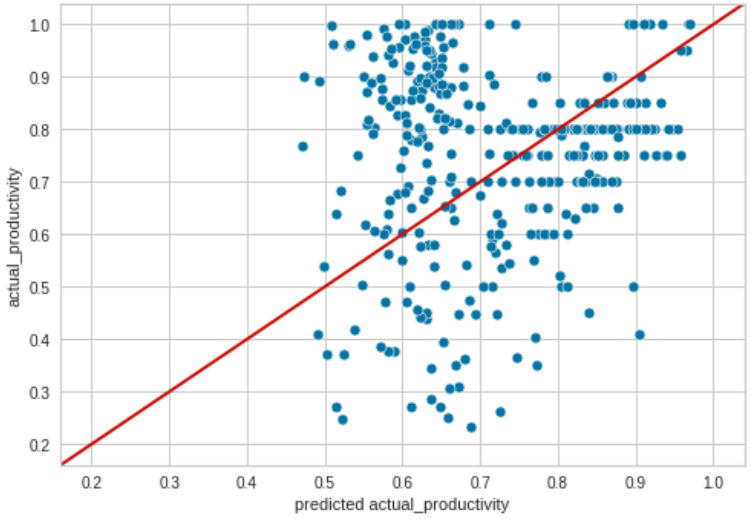

Figura 8: predicciones de un modelo de regresión ridge

En la figura 9, se puede observar que realmente da muy malos resultados en el test: se puede constatar cómo las muestras se encuentran dispersas y anárquicamente alejadas de la diagonal, que representaría la región donde idealmente las muestras deberían alinearse (ello implicaría alcanzar una precisión del 100%). Además, vale la pena destacar la alineación de ciertas muestras que, en definitiva, provocarían ese error de *test*.

Teniendo en cuenta lo dicho, este modelo no es adecuado para resolver el presente problema.

SVM con kernel cuadrático

Su exploración hiperparamétrica corresponde a la siguiente:

- **C**, para cuyos valores hemos explorado un rango entre $[-5, 5]$, tratando de tomar muestras cada 0.05 unidades. Dicho rango se ha elegido teniendo en cuenta el tamaño (relativamente reducido) de datos con que trabajamos, además del hecho de garantizar una exploración adecuada de la regularización aplicable a los datos.
- **degree**, para cuyo valor se ha fijado 2. Este hiperparámetro es precisamente el que garantiza que la máquina de soporte vectorial tenga kernel cuadrático.
- **gamma**, cuyos valores probamos enteramente debido a que pretendemos explorar exhaustivamente aquellos los hiperparámetros con valores prefijados.

Algo a tener en cuenta es el hecho que, para los 5 mejores modelos, los hiperparámetros escogidos son parecidos (*C* oscila en valores alrededor de 1 y 3, mientras que para *gamma* siempre se elige *auto*).

El modelo entrenado a partir de dicha exploración hiperparamétrica resulta tener malos resultados: el mejor de ellos parece alcanzar un error de entrenamiento de 0.287. Además, su error de test alcanza un valor de -0.221. La visualización de la relación entre valores reales y predicciones es la siguiente:

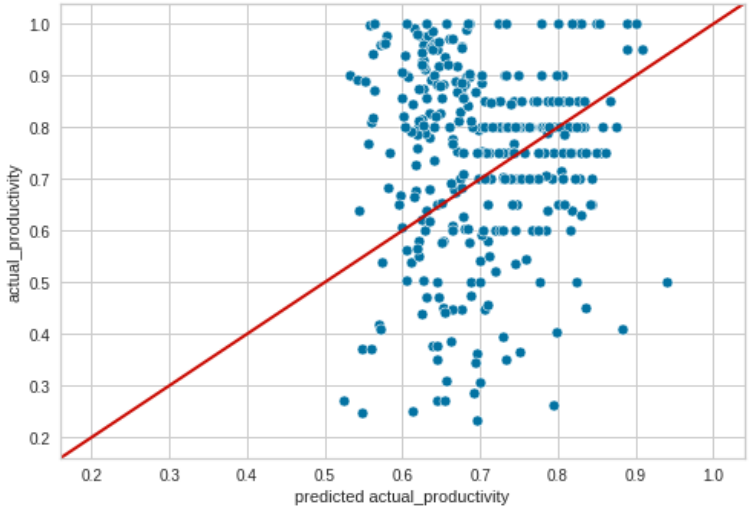

Figura 9: predicciones de un modelo de SVM con kernel cuadrático

Tal y como se puede apreciar en la figura, las instancias se encuentran dispersas y para nada alineadas sobre la diagonal, lo que apunta a un fracaso total del modelo a la hora de realizar predicciones de calidad. Además, de nuevo observamos la aparición de esas franjas horizontales, si bien esta vez son menos espaciadas que en el anterior modelo.

Finalmente, podemos afirmar con rotundidad que los modelos lineales aquí explorados no son capaces de responder adecuadamente ante el problema aquí tratado.

6. Modelos no lineales

De nuevo, teniendo en cuenta el análisis del TSNE, la asunción inicial es que los modelos no lineales tampoco van a terminar de ser capaces de responder adecuadamente al problema que traemos entre manos. Sin embargo, vamos a tratar de comprobar si esto es cierto. Los modelos que se verán a continuación corresponden a una SVM con kernel RBF, un random forest y un perceptrón multicapa.

SVM con kernel RBF

La exploración hiperparamétrica de este modelo es igual a la efectuada en la anterior máquina de soporte vectorial (con la excepción que nos desaparece el hiperparámetro *degree*).
Con esto, procedemos a entrenar el modelo, y como resultado obtenemos un error de entrenamiento de 0.284, el cual es igual de pésimo que en los anteriores casos. Vale la pena destacar que, de nuevo, para los 5 mejores modelos, los valores elegidos para los hiperparámetros explorados son muy similares (*C* oscila entre 1 y 3 mientras que *gamma* siempre elige la opción “auto”). Además, en cuanto a su error de test se refiere, este es de -0.603, cuyo valor sería el peor obtenido hasta ahora. Veamos exactamente cómo luce la representación de las predicciones respecto los valores reales:

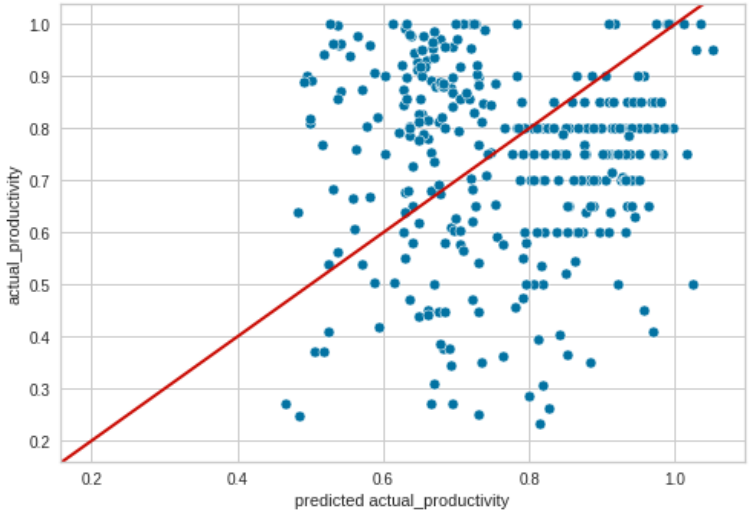

Figura 10: predicciones de un modelo de SVM con kernel RBF

Tal y como se observa, las muestras se alejan de la diagonal de formas parecidas a las visualizadas anteriormente, y de nuevo observamos la aparición de las franjas horizontales que hasta ahora venimos visualizando. Lo que de ello se deriva es que la complejidad del conjunto de datos, hasta ahora, está siendo difícil de atacar.

Random Forest

Este tipo de modelo, a diferencia de los que venimos viendo, tiene un claro punto a favor, y es que su espacio de hiperparámetros a explorar es más elevado y más significativo para el entrenamiento del modelo. Veamos cuáles vamos a explorar:

- **n_estimators**: sabiendo que el número por defecto es 100, y conociendo el tamaño del conjunto de datos con que trabajamos, creemos adecuado explorar valores en $\{5, 10, 20, ..., 100\}$ con tal de permitir al modelo alcanzar cierto equilibrio entre la granularidad y la generalización explorada en en el entrenamiento sobre los datos.
- **criterion**: elegimos explorar todos los valores prefijados para esta variable, con tal de garantizar que no queda un espacio de posibles combinaciones de hiperparámetros sin comprobar.
- **max_depth**: sabiendo que su valor por defecto es *nulo*, decidimos probar valores entre 2 y 80, incluyendo también el valor *nulo*, con tal de verificar si el modelo rinde mejor cuando no hay posibilidad de sobreespecialización (a la vez que se explora el forzamiento de la generalización que viene implícito con ello).
- **min_samples_leaf**: sabiendo que el valor por defecto es 1, miramos de explorar valores por encima del mismo (en el conjunto $\{1, 2, 3, 4, 5, 10\}$) con tal de revisar si, forzando cierta generalización sobre los datos, el modelo se desempeña mejor.

Con esta exploración, el mejor modelo entrenado alcanza un error de entrenamiento de 0.452, el mejor hasta la fecha, y un error de *test* de $-\ 0.655$, que es incluso peor que el del modelo anterior. Con ello, claramente vemos como la complejidad del conjunto de datos es tal que ni siquiera los modelos mejor entrenados hasta ahora son capaces de realizar predicciones aceptables. Para terminar de corroborarlo, veamos la representación de las predicciones respecto los valores reales:

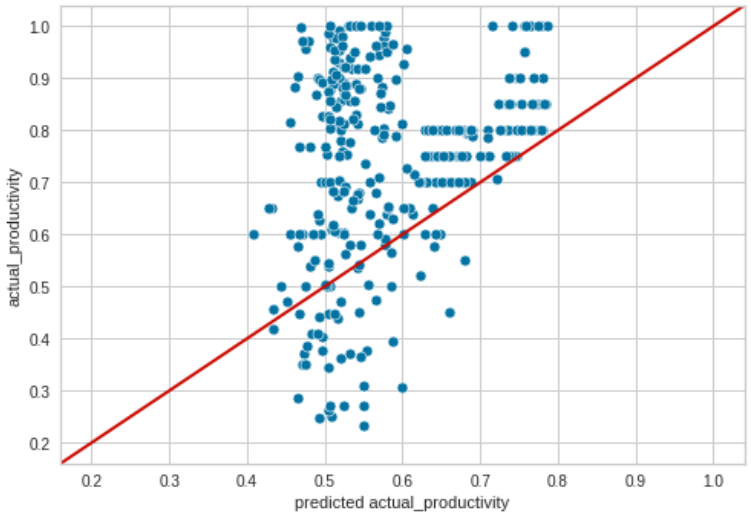

Figura 11: predicciones de un modelo de random forest

Tal y como podemos apreciar, los datos no solamente no se alejan de la diagonal, sino que forman franjas y concentraciones como las que veníamos visualizando hasta ahora. En definitiva, el presente modelo no es mejor que los anteriores, salvo para el conjunto de *train*.

Perceptrón Multicapa

Este tipo de modelos tienen una complejidad inherente, motivo por el cual son costosos (tanto temporalmente como computacionalmente) de entrenar. Es precisamente por esta complejidad que esperamos que el perceptrón sea capaz de realizar predicciones de calidad del presente conjunto de datos.
El modelo que aquí presentamos, como se comentó en la sección 2, se encuentra inspirado y expande sobre otra red neuronal ya diseñada para este conjunto de datos [2] y cuya forma originaria es la siguiente:

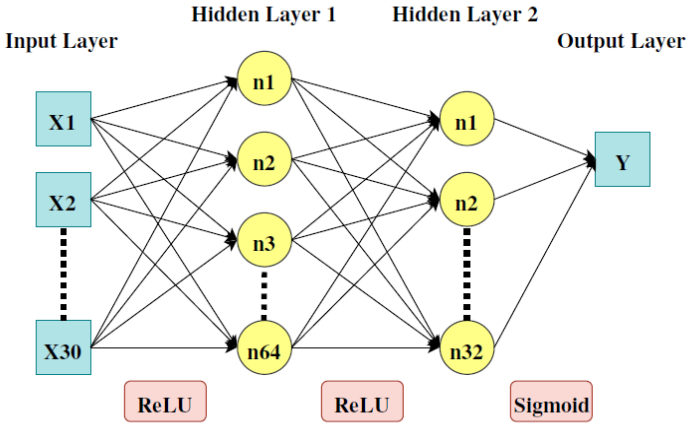

Figura 12: MLP original

De acuerdo con la estructura visualizada, hemos tratado de probar a modificar todas aquellas componentes susceptibles de serlo:

- El tamaño de la capa de entrada y de salida: de acuerdo con las orientaciones originales del diseño del perceptrón, hemos estimado adecuado garantizar que la capa de entrada tenga tantos nodos como variables tengamos en el dataset, mientras que la capa de salida se mantenga con 1 único nodo.
- El tamaño de las capas ocultas: partiendo del hecho que hemos cambiado los tamaños por defecto de dichas capas (cuyos valores serían ahora 26 y 13), hemos optado por probar 3 combinaciones de tamaños distintas: $(78, 52), (52, 26), (26, 13)$. Se ha elegido la cantidad de 2 capas debido a que, según se indica [6], es una cantidad que permite al modelo captar aproximadamente (con mucha o poca precisión) la naturaleza de un conjunto de datos dado.
- Las funciones de activación: manteniendo la combinación (*ReLU*, *sigmoid*), hemos optado por añadir (*sigmoid*, *linear*) y (*linear*, *sigmoid*) a las posibles combinaciones de funciones de activación del modelo, con tal de explorar distintas combinaciones y su potencial efecto sobre el modelo.
- El tamaño de *batch*: teniendo en cuenta que el conjunto de entrenamiento cuenta con alrededor de 830 instancias, hemos optado por probar unos valores de batch de $\{25, 50, 100\}$. Con ello, tratamos de comprobar cuál es la respuesta del modelo ante las maneras que este tiene de entrenarse (si con más o menos instancias por sub-entrenamiento rinde mejor o peor).
- El número de *epochs*: dado el tamaño y complejidad del conjunto de datos, juntamente con la complejidad inherente al modelo, consideramos conveniente explorar un conjunto de valores de $\{50, 100, 200\}$ para *epochs*. Precisamente por

esto nos interesa entrenar iterativamente el modelo una cantidad de veces equilibrada, esto es, una cantidad tal que garantice que el modelo es entrenado adecuadamente a la vez que evita consumir tiempo innecesariamente para ello.

Visto esto, y teniendo en cuenta que el modelo es entrenado con la función de *MSE* como función de pérdida, el error de entrenamiento del mejor modelo es de 0.517, juntamente con un error de test de 0.075, los cuales siguen siendo igual de pésimos. En esta línea, visualicemos las predicciones contra los valores reales:

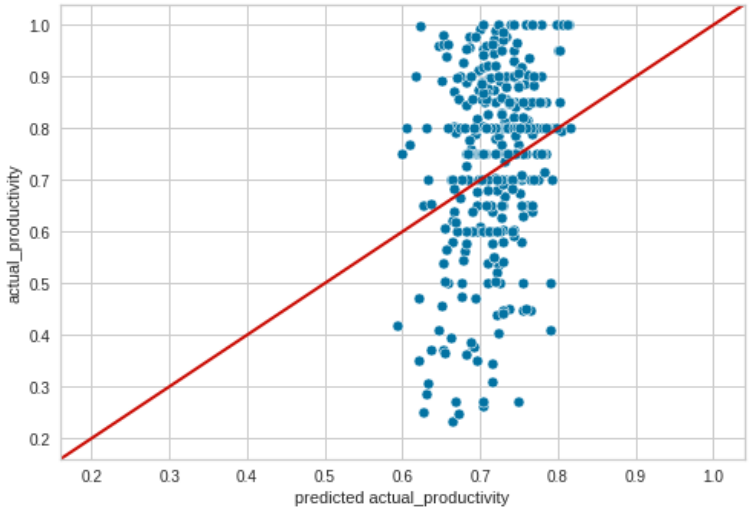

Figura 13: predicciones de un modelo MLP

Podemos constatar que, si bien es cierto que las muestras siguen mostrándose alejadas de la diagonal, este modelo consigue concentrar una cantidad relativamente mayor de instancias cerca de esta, lo que propicia un error de *test* positivo. Sin embargo, esto no es suficiente para poder afirmar que el modelo es capaz de realizar predicciones aceptables sobre los datos con que trabajamos.

7. Modelo definitivo

Todos los modelos se han desempeñado mal para hacer predicciones adecuadas sobre el problema con que trabajamos, y casi ninguno de ellos ha sido capaz de alcanzar un error de *test* positivo, con lo cual no son recomendables para efectuar ningún tipo de predicción sobre el conjunto de datos aquí tratado.

Dicho esto, tras todas las visualizaciones efectuadas, y teniendo en mente estos resultados, optamos por el perceptrón multicapa como el mejor modelo: su inherente complejidad es la que le ha permitido adaptarse lo suficiente como para llegar a alcanzar las mejores métricas de error.

Dicho esto, es evidente que no es un modelo aceptable, puesto que su error de generalización, expresado según la métrica de R^2, a pesar de ser positivo, tiene casi la misma calidad de respuesta que una línea horizontal y efectuar las predicciones de acuerdo con esta. Por ello, su error de generalización no es aceptable bajo ninguna circunstancia, y no nos habilitaría a ponernos siquiera en predisposición de considerar que pueda efectuar predicción alguna.

8. Interpretabilidad de los modelos

Con tal de entender exactamente a qué se deben las predicciones realizadas, hemos usado métodos de interpretabilidad para sacar información acerca de la importancia que guardan las variables para cada uno de estos mediante *permutation importance*. Lo que en esta sección va a presentarse es una comparativa de las variables elegidas por los modelos lineales y no lineales, para posteriormente explicar un modelo, a manera de contraste, de cada tipo.

Relevancia de los atributos

Visualicemos primero el resultado de la *permutation importance* para ambos tipos de modelos:

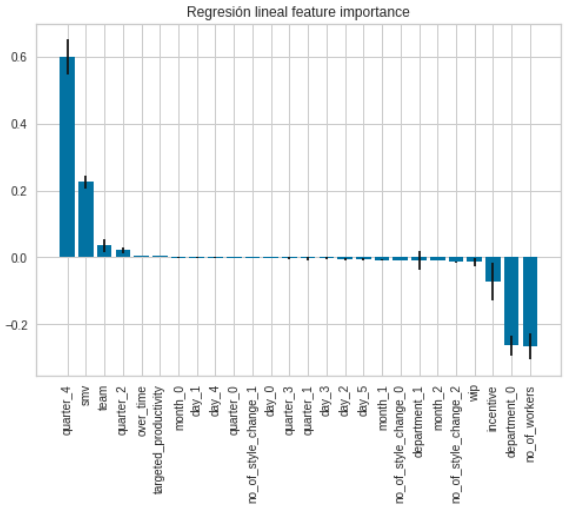

Figura 14: permutation importance de la regresión lineal

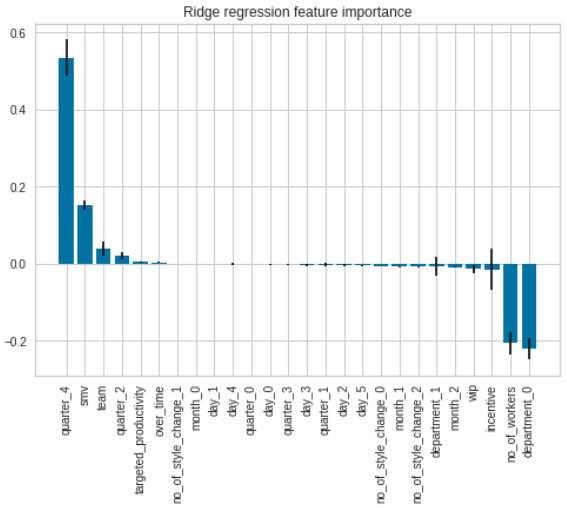

Figura 15: permutation importance de la regresión ridge

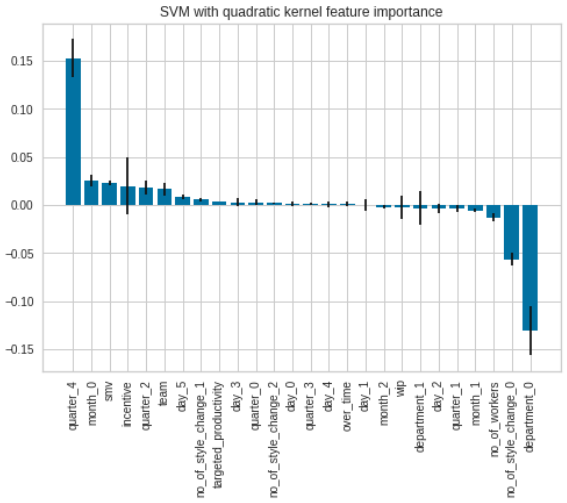

Figura 16: permutation_importance de la SVM con kernel cuadrático

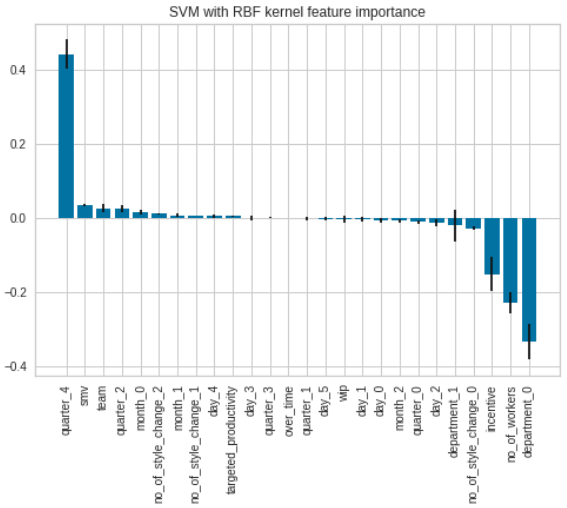

Figura 17: permutation importance de la SVM con kernel RBF

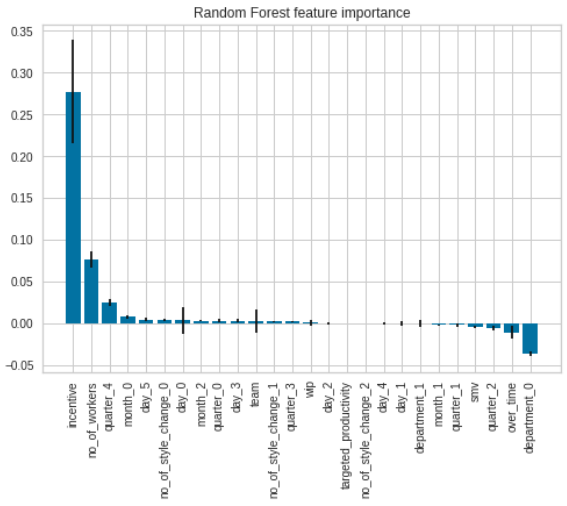

Figura 18: permutation importance del random forest

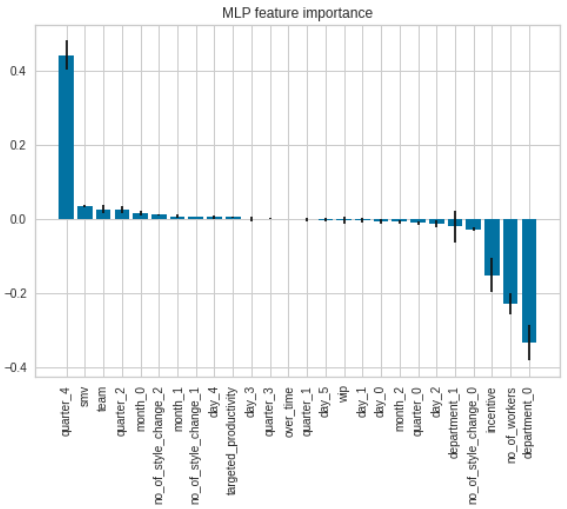

Figura 19: permutation importance del perceptrón multicapa

Por un lado, observamos que en los modelos lineales (figuras 14, 15 y 16) se capta la variable *quarter_4* como la más importante, y mientras que para ambas regresiones las siguientes son las mismas (más o menos en el mismo orden), la máquina de soporte vectorial presenta ligeros cambios (véase el caso de *incentive*). Además, también se puede observar que, para las variables de importancia negativa, el parecido es abrumador para las regresiones (comparten los mismos atributos, con ligeras diferencias en su orden), mientras que para la SVM, de nuevo, aparecen y desaparecen variables (como es, de nuevo, el caso de *incentive*, que cobraría importancia ante *no_of_style_changes*, que pasaría a tener importancia negativa). Adicionalmente, cabe destacar la desviación estándar de algunas de las variables, puesto que en algunos casos esta hace peligrar su importancia.

Por otro lado, para los modelos no lineales (figuras 17, 18 y 19) vemos algunas particularidades: tanto el modelo del MLP como la SVM comparten virtualmente la misma *permutation importance*, que también parecería coincidir, a grandes rasgos, con aquella visualizada para las regresiones. Visto esto, el modelo de random forest sería el único que parecería tomar en consideración variables completamente distintas a las que venimos viendo (considera importantes variables como *incentive* o *no_of_workers*, que hasta ahora casi todos los modelos consideraban como desechables).

En definitiva, parece que, generalmente, todos los modelos juzgan las variables de la misma forma, y parecen dar más importancia al equipo o la semana en que se realiza el trabajo que el incentivo monetario o la productividad deseada. Esta visualización nos permite elegir la regresión lineal y el random forest como representantes de sus tipos de modelos, tanto por su *permutation_importance* como por su calidad (dado que todos los modelos son malos, no importa cuáles elijamos entre sí).

Explicación de un modelo de cada tipo

Para tener una comprensión más adecuada del funcionamiento de los modelos, estudiaremos qué efecto tienen sobre el conjunto de datos de *train*. En cuanto al estudio efectuado, para ambos modelos exploramos valores atípicos, valores extremos y valores máximos y mínimos para el valor de salida de la variable objetivo. Consideramos que esta selección de instancias de ejemplo será representativa del funcionamiento de los tipos de

modelos aquí estudiados, puesto que constituyen ejemplos de instancias interesantes a ser estudiadas. Adicionalmente, vale la pena destacar que, para cada conjunto de datos explorado, estudiamos tanto aquellos con mejor error (esto es, aquellos en que las predicciones y el valor de la variable objetivo sean virtualmente iguales) como aquellos con peor error (cuyo caso sería el alternativo al descrito). Ello se hace estudiando los valores que tienen una diferencia por debajo de $0,05$ (es decir, los valores son prácticamente iguales) o por encima de una cota variable (cambia según el error máximo registrado para el conjunto de datos explorado). También cabe señalar que, para intentar explicar el modelo, intentamos buscar patrones decisionales de este sobre las variables para los mismos grupos de ejemplos mencionados anteriormente, tratando así de detectar qué tipo de valores han de tener para que se predigan bien, o los valores que pueden confundir al modelo, haciendo que se prediga mal. Finalmente, consideramos que la explicación debe hacerse sobre el conjunto de entrenamiento, puesto que serían precisamente los que conoceríamos y que, por lo tanto, deberíamos poder explicar.

Regresión lineal

Para poder estudiar ejemplos y sus debidos valores, empezamos visualizando el boxplot asociado a sus predicciones:

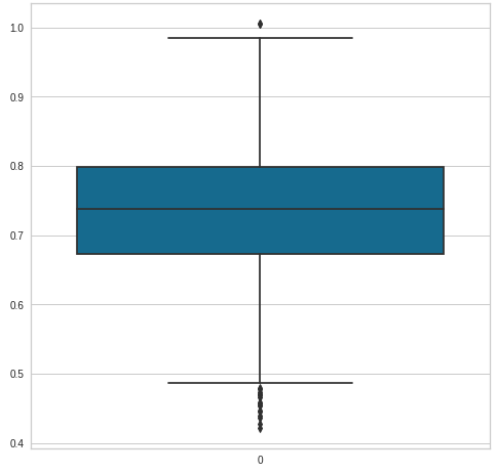

Figura 20: boxplot de las predicciones de la regresión lineal

Tal y como se observa, tenemos presencia de valores atípicos para ambos laterales, por lo que empezaremos por estos. Lo primero que cabe destacar es el desequilibrio existente, dado que tenemos más valores atípicos para la parte inferior que para la superior. Si visualizamos los superiores, vemos que los valores de sus atributos son prácticamente idénticos, y ambos precisamente corresponden a instancias que son bien predichas (la diferencia entre la predicción y el valor real es aproximadamente 0). Cabe destacar que, a pesar de ello, hay que tener en cuenta que solo son 2 instancias, por lo que no podemos generalizar estas afirmaciones (ni mucho menos). Dicho esto, para los valores atípicos inferiores encontramos una mayor complejidad: la diferencia entre predicción y valor real de la variable objetivo llega a alcanzar una diferencia de 2 décimas, lo que es mucho si se tiene en cuenta que los valores se encuentran en el rango [0,1]. Si se exploran las instancias con mejor y peor error de este subconjunto, podremos ver que no hay un patrón claro mediante el cual el modelo prediga correctamente algunas de ellas y otras no.

Respecto a los valores extremos, encontramos una mayor cantidad de instancias que en los casos anteriores, lo que implica más disparidad de resultados: tanto para los valores extremos superiores como inferiores, encontramos que la diferencia entre predicciones y valores reales llega alcanzar entre 4 y 5 décimas de diferencia, lo que apunta a una variabilidad tan grande en el error de las predicciones que difícilmente, mirando los valores de sus atributos, podemos sacar algo en claro. Si miramos tanto aquellas instancias con mejor error que con peor error, encontramos una variabilidad en los valores de sus atributos que nos hace complicado (si no imposible) encontrar un claro patrón decisional mediante el cual el modelo prediga bien (o no).
Finalmente, enfocándonos ahora en los valores máximos y mínimos asociados al valor de salida (esto es, en el valor de la variable objetivo), exploramos valores de productividad por encima de 0.999 y por debajo de 0.38, limitando así la cantidad de instancias que se toman. Estas cotas se eligen teniendo en cuenta tanto la corrección de valores realizada en el preprocesamiento (con seguridad tendremos valores de 1) como las visualizaciones realizadas (sabemos que no tenemos productividad por debajo de 0.2, por ejemplo). Dicho esto, si se hace dicha elección, podemos obtener alrededor de 20 instancias para cada tipo de valor, y si observamos su diferencia entre predicción y valor real, encontramos que, para ambos casos, esta alcanza las 2 décimas de diferencia, y tanto la exploración para las mejores predicciones como para las peores nos lleva a un callejón sin salida: no hay patrones decisionales claros.
Para los ejemplos visualizados, no hemos sido capaces de encontrar, tanto para instancias bien predichas como mal predichas, relación alguna entre variables. Además, no existe relación clara entre instancias bien/mal predichas y valores de las variables, tanto para las relevantes como para las irrelevantes (*quarter_4*, *smv*, *no_of_workers*, etc.). En definitiva, todo esto viene derivado, en última instancia, de la complejidad de los datos, complejidad que impide al modelo adaptarse a los datos y realizad buenas predicciones.

Random forest

Para el presente modelo, volvemos a visualizar el boxplot de sus predicciones

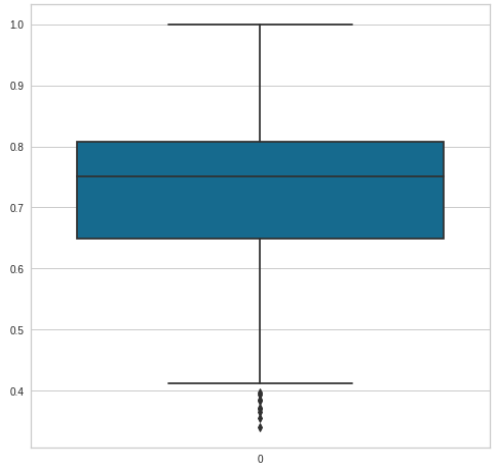

Figura 21: boxplot de las predicciones del modelo de random forest

Tal y como se observa, estas solo presentan 9 valores atípicos en la parte inferior de la figura, valores en los cuales las predicciones oscilan entre 0.015 y 0.134, lo cual supone una diferencia de 1 décima, y que no implica una variabilidad muy grande. Si miramos las

instancias con mejor error, da la casualidad de que en 3 de las 4 tienen el mismo valor para las dos variables que más importancia tienen para el modelo, que son *incentive* y *no_of_workers*. En cambio, para las que tienen los errores peores, hay 2 de las 3 que siguen ese patrón, lo que implicaría que dicho patrón no sería adecuado para identificar instancias bien/mal predichas.

Para el grupo de valores extremos, a pesar de que hay 209 valores en los superiores y 200 en los inferiores, aumenta ligeramente la varianza con respecto al error. Ahora este oscila entre 0 y 0.154, lo que supone una diferencia de 1.5 décimas entre la predicción y el valor real. En cambio, al representar 10 valores con las predicciones que más se acercan del valor real y otras 10 con las que más se alejan, no se ha podido observar ningún tipo de patrón en las variables, ya que hay una variabilidad muy grande entre las variables que el modelo considera muy importantes: a pesar de que las variables visualizadas solamente contienen 20 de las 409 totales, la variabilidad de *incentive* es de casi 1 unidad y la de *no_of_worlers* es de 0.6, lo cual es mucho si consideramos que los valores están normalizados en el rango [0,1].

Para el grupo de valores máximos y mínimos de salida, (esto es, en el valor de la variable objetivo), exploramos valores de productividad en el mismo rango que para el anterior modelo. Ahora, estos oscilarían entre 0 y 0.138.

No se puede observar ningún tipo de patrón, ya que las variables tienen mucha variabilidad, sobre todo aquellas que el modelo considera más relevantes, como *incentive*, con prácticamente 1 unidad de diferencia (va entre 0 y 0.96 aproximadamente) y *no_of_workers,* con 6 décimas de diferencia (entre 0.06 y 0.65). En definitiva, al igual que con la explicación del modelo lineal, todo esto viene derivado de la complejidad de los datos.

9. Conclusiones

Lo primero que deberíamos comentar son los resultados obtenidos: ningún modelo ha sido capaz de alcanzar una calidad aceptable, esto es, de acuerdo con los errores de entrenamiento y de *test*, ninguno de los modelos es capaz de alcanzar un mínimo de calidad, ya que no consiguen adaptarse bien al conjunto de datos. Si además tenemos en cuenta el resultado de los análisis del PCA y del TSNE, vemos que esa complejidad a la hora de predecir la variable objetivo es real. Adicionalmente, recuperando la interpretabilidad de los modelos, cabe añadir que estos, a nuestro parecer, apuntan en direcciones erróneas, dado que parecen captar la importancia en variables que, según nuestra intuición, no se corresponden con el problema aquí planteado (¿qué sentido tendría, a priori, que *team* tenga más importancia que *targeted_productivity* o *incentive* si estamos tratando de predecir la productividad real?); lo que sí es cierto es que al menos parece que hemos captado una posible línea de razonamiento acerca de por qué los modelos responden de la manera que lo hacen.

En cuanto a nuestro juicio acerca de los resultados, el proyecto ha fracasado: los modelos no han sido capaces de adecuarse suficientemente bien a la partición de entrenamiento, su error de *test* ha sido estrepitosamente malo, y para el que ha dado mejores resultados, sigue siendo más inapropiado que asumir la media de los valores de la variable objetivo. Cabe añadir que tenemos nuestras dudas acerca de la naturaleza de las variables y del análisis realizado: hemos asumido que eliminar variables temporales (tales como *day* o *quarter*, por ejemplo) nos haría perder información relevante que pudiera ayudarnos a realizar las predicciones correctamente (por ejemplo, es posible que la productividad durante alguno de los primeros meses sea más elevada debido a la época de festivos), pero desconocemos si esto solo hace que apuntalar la complejidad de los datos; tampoco sabemos si deberíamos haber realizado la partición (y por ende el entrenamiento) tratando los datos como una serie temporal (hemos asumido que esta componente de temporalidad podría incluso aumentar la complejidad de los datos); finalmente, también hay que tener en cuenta que el preprocesamiento no es único (por ejemplo, para los outliers nuestro tratamiento se ha basado en asumir que los valores eran correctos y, por lo tanto, hemos tratado de suavizarlos, sustituyéndolos por los valores extremos de sus cuartiles). En definitiva, hay cuestiones que estudios complementarios podrían tratar de dilucidar con lo que se ha conseguido en este trabajo.

En cuanto a valoraciones personales, honestamente afirmamos que, basándonos en el estudio preliminar, no esperábamos que un conjunto de datos de tales características (buena proporción entre cantidad de variables y número de instancias) fuera a ser tan complejo y problemático. Ello nos advierte que, no por tratarse de conjuntos relativamente pequeños de datos, los problemas se vuelven más sencillos o factibles de resolver.

También estimamos conveniente añadir que el presente trabajo ha enfocado el problema como uno de regresión, y que ello no tiene por qué implicar que esta complejidad que venimos comentando deba ser naturalizada, puesto que podría ser producto del tipo de tarea que tratamos de resolver: nosotros buscábamos realizar predicciones concretas, tratando así de dar una cifra exacta acerca de la productividad que cada equipo, en cada departamento, en una fecha determinada y que bajo unas ciertas condiciones de producción tendría. En definitiva, no debemos desestimar tratar de trabajar este dataset desde la clasificación: tal y como en los estudios previos hemos comentado, ya ha habido experimentación acerca de cómo crear clases para la tarea. Lo que nosotros proponemos

como potencial extensión, inspirandonos en la idea del estudio citado [1], es la creación de dos clases de la siguiente forma: la clase “0” si su *targeted_productivity* < *actual_productivity*, o alternativamente la clase “1” en caso contrario.
Finalmente, como posible mejora podríamos intentar resolver el origen del 42 % de missing values presentes en *wip* hablando con la persona que recogió los datos, y/o tratar de tomar otras decisiones en el preprocesamiento con tal de ver si mejora. También, se puede probar con algún otro modelo que no se haya considerado y que se crea que pueda desempeñarse mejor. Estos posibles cambios puede que no mejoren nada los resultados obtenidos debido a las visualizaciones del PCA y el TSNE, pero hay que tener en consideración que el PCA tenía una fiabilidad de menos del 40%, y que no hayamos obtenido buenos resultados no quiere decir que no puedan obtenerse.

10. Bibliografía

[1] Cerit, E. (2021) *Notebook-Garment Classification*, *Kaggle*. Kaggle. Available at: https://www.kaggle.com/code/eminecerit/notebook-garment-classification (Accessed: January 2, 2023).

[2] Imran, A. A., Amin, M. N., Islam Rifat, M. R., & Mehreen, S. (2019). Deep Neural Network Approach for Predicting the Productivity of Garment Employees. 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT).

[3] Turney, S. (2022) *What is kurtosis?: Definition, Examples & Formula*, *Scribbr*. Scribbr. Available at: https://www.scribbr.com/statistics/kurtosis/ (Accessed: January 2, 2023).

[4] API Reference - *Keras Documentation: Keras API reference*. Available at: https://keras.io/api/ (Accessed: January 2, 2023).

[5] *Textile Market Size, trends and Global Forecast to 2032* (no date) *The Business Research Company*. Available at: https://www.thebusinessresearchcompany.com/report/textile-global-market-report (Accessed: January 2, 2023).

[6] Heaton, J. (2008). *Introduction to neural networks with Java*. Heaton Research, Inc.