Trabajo Fin de Máster

Máster en Ciencia de Datos e Ingeniería de Datos en la Nube

Detección de desviaciones para corrección de colas de dosificación en procesos automáticos de fabricación de pienso de alimentación animal.

Autor: José Luis Casado Valero

Tutor: Tutor nombre

Co-Tutor: Co-tutor nombre

Septiembre, 2021

Dedicado a mi familia y a todos aquellos…….

Declaración de Autoría

Yo, José Luis Casado Valero con DNI 74512698-N, declaro que soy el único autor del trabajo fin de grado titulado "Detección de desviaciones para corrección de colas de dosificación en procesos automáticos de fabricación de pienso de alimentación animal" y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a.....

Fdo: José Luis Casado Valero

Resumen

En la industria agroalimentaria, en concreto, fábricas de producción de pienso animal, existen gran cantidad de procesos susceptibles de ser automatizado, la maquinaria utilizada no es de muy alta precisión en cuanto a las mediciones, por lo que los errores en cuanto a stock, dosificaciones, estimaciones, etc. son muy habituales.

Una fábrica de pienso en líneas generales es perecida a un "Telepizza", hay una recepción de materias primas (cereales, trigo, soja, grasas, líquidos, …) las cuales son almacenadas en silos (depósitos), mediante una formulas o recetas se marcan los ingredientes que cada pienso compuesto y son los procesos automáticos los que dosifican, mezclan y tratan los ingredientes para formar piensos compuestos que son expedidos a las granjas para el consumo animal.

Dentro de todo este ciclo de vida del pienso hay multitud de procesos automáticos, en muchos de ellos se utilizan diferentes sondas y sensores, pero en algunos de ellos no se disponen de esos elementos por lo que aumenta el nivel de incertidumbre y con esto el número de errores en muchos de las decisiones en tiempo real que hay que tomar en la automatización.

Uno de los procesos más importantes y que se produce en diferentes fases de la fabricación es la dosificación de materias. Los silos comentados anteriormente tienen en su parte inferior una rasera / compuerta o un rosca sinfín que al ser activadas dejan pasar la materia, esta cae por gravedad en una báscula de pesaje que nos indica la cantidad dosificada. Entre el elemento mecánico (rasera o rosca) activado y la báscula hay una

distancia, por lo que una vez que se para o cierra el elemento sigue cayendo materia a la bascula durante un tiempo, lo cual nos provoca desviaciones en las mediciones.

Para corregir este fenómeno se utilizan lo que denominamos “colas de caída”, que es la cantidad de materia que cae desde que desactivamos el elemento hasta que se estabiliza la báscula. Dentro del proceso automático lo que se hace es desactivar el elemento mecánico un poco antes de que la báscula marque la cantidad deseada, en concreto el valor de esa “cola de caída” para intentar que cuando la báscula se estabilice marque la cantidad deseada.

Las colas de caída son muy variantes, dependen mucho de la báscula, del elemento dosificador, de la materia, lo que se ha dosificado antes, etc. incluso de la temperatura, humedad. El calculo actual de colas se basa en algunas de las variables anteriormente mencionadas en las últimas dosificaciones realizadas.

El sistema del que se dispone actualmente es correcto y funciona aceptablemente, pero en ciertas circunstancias el modo de calculo es insuficiente y se producen ciertas desviaciones que llevan a errores en las dosificaciones, debido a ciertas variables que influyen en la dosificación y de las que no se tiene información, como la temperatura, humedad, etc.

La idea de este trabajo fin de máster es, teniendo información de las dosificaciones realizadas en los últimos años en una fábrica, en torno a los 2 millones, realizar un trabajo de limpieza, estudio, creación de características y obtención de un modelo que sea capaz de predecir esas desviaciones esporádicas que se producen y con ello corregir la cola de caída utilizada y evitarlas.

Agradecimientos

A todas las personas que se lo merecen.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Introducción

Como ya sabemos, el proceso de dosificación de materia utilizado en una fábrica de pienso de alimentación animal se realiza mediante la activación de elementos mecánicos que dejan pasar la materia almacenada en un silo/depósito dejándola caer en una báscula de pesaje, la cual nos indica el peso de la materia dosificada.

El problema en este tipo de procesos es que entre el elemento dosificador y la báscula hay cierta distancia, por lo que desde que el elemento se desactiva hasta que bascula deja de variar su peso pasa cierto tiempo. La cantidad de materia que cae en ese periodo de tiempo es lo que denominamos “cola de caída”.

Si no contemplamos este fenómeno, existirán errores en todas las pesadas realizadas, para corregirlo, lo que se hace es desactivar el elemento dosificador cuando nos falte el valor correspondiente de la “cola de caída” para llegar a nuestro objetivo y así la bascula marcará el peso deseado cuando se estabilice.

El sistema utilizado en la actualidad tiene en cuenta una serie de variables comunes a todas las dosificaciones, como son: materia a dosificar, silo del que se dosifica, materia

que se quiere obtener, etc. … y se basa en las ultimas pesadas, o mejor dicho en la cola que hubiesen necesitado las ultimas pesadas para esas variables para calcular la nueva cola para la siguiente pesada.

Estas colas al tener en cuenta siempre las ultimas pesadas, se van actualizando después de cada una de las pesadas por lo que van variando con el tiempo. Para el caso de las primeras pesadas que no hayan sido contempladas con anterioridad se utilizan unas colas por defecto que nos sirven para iniciar los datos almacenados para ese conjunto de variables.

Lo bueno de este sistema es que aprende muy rápido y los resultados obtenidos son bastante aceptables, pero en ciertas circunstancias hace que se produzcan desviaciones muy elevadas en las pesadas ya que no tienen en cuenta ciertos factores por los que se pueden ver afectadas.

1.2 Objetivos

El objetivo de este trabajo no es en si modificar el método de cálculo de colas, ya que como he comentado, su funcionamiento es aceptable, sino obtener un modelo de predicción que lo complemente.

La idea es, basándonos en los datos recopilados de las dosificaciones realizadas hasta la fecha, poder predecir las desviaciones que se producen por no tener en cuenta otros factores no contemplados en el cálculo de colas.

Hay muchos factores que pueden influir y que nos pueden ayudar a refinar el sistema y predecir cuando una dosificación puede sufrir una desviación, como por ejemplo la fecha en la que se produce, la hora, el día de la semana, la pesada o pesadas que se han realizado anteriormente, temperatura, humedad, etc.

El modelo resultante de este trabajo será utilizado para predecir con anterioridad a cada una de las pesadas la desviación que se va a producir según las circunstancias y el

momento en el que se va a realizar, dicha desviación complementará / corregirá a la cola calculada con el método actual, y conseguiremos reducir las desviaciones que se producen en la actualidad.

1.3 Estructura del proyecto

Este trabajo será dividido en diferentes etapas, que básicamente son las etapas de un proyecto de machine learning, aunque anteriormente nos centraremos en explicar el significado de los datos de partida con el fin de comprender mejor el problema y el objetivo que pretendemos conseguir.

A continuación, haremos un análisis exploratorio de los datos, identificando lo diferentes tipos de datos, las variables numéricas, las variables categóricas, etc. Realizaremos una limpieza de estos, buscando valores perdidos, outliers e intentaremos crear ciertas características que pueden ayudar a caracterizar el problema.

Es siguiente paso será realizar un preprocesamiento de los datos, imputando valores perdidos, estandarizando y escalando variables numéricas, etc…

Entrenaremos diferentes modelos y realizaremos su validación correspondiente para quedarnos con el que mejores resultados obtengamos y que pueda ser pasado a producción dentro de nuestros procesos.

Capítulo 2

Presentación de los datos

2.1 Introducción

En este apartado describiremos la información de partida para la obtención del modelo final con el fin de entender la naturaleza del problema y poder desarrollar diferentes ideas o estrategias en las fases posteriores de análisis y preprocesamiento.

2.2 Fuente de datos

Los datos de los que disponemos son extraídos de la propia aplicación que gestiona los procesos de automatización de la fabrica de piensos. Dentro de la amplia legislación que rige la industria agroalimentaria, existe una estricta normativa relativa a la trazabilidad. Este tipo de procesos deben tener la capacidad seguir el rastro a toda materia prima que forme parte de este desde que es adquirida hasta que es consumida.

Como hemos comentado en una fabrica de pienso existen gran cantidad de procesos de transferencia y transformación de materia, y todos ellos deben quedar registrados para una posible consulta de trazabilidad. En el caso concreto que nos ocupa, las dosificaciones de materias primas, se produce una transferencia de materia y a su vez un mezclado de esta, formándose una materia compuesta. Cada una de las dosificaciones es registrada y la información que obtenida se describe en la siguiente tabla. **Tabla 2.2.1.** Muestra de datos de partida basados en dosificaciones anteriores

Tabla 2.2.1. Muestra de datos de partida basados en dosificaciones anteriores

Nombre columna	Ejemplo	Descripción
Fecha_inicio	2013-08-14 16:51:48	Fecha de inicio de la dosificación
Fecha_fin	2013-08-14 16:52:39	Fecha de fin de la dosificación
Cantidad_solicitada	235.026	Cantidad objetivo a dosificar
Cantidad_dosificada	236.5	Cantidad real dosificada
Mezcla	1	Nº de mezcla dentro de la fabricación
Pesada	0	Nº de pesada dentro de la mezcla
Peso_inicial	0.0	Peso que marca la bascula antes de inicar la dosificación
Manual	0	Booleano que indica si la dosificación ha sido manual o automática
Materia_origen	13	Código de la materia origen.
Materia_destino	66	Código de la materia destino
Id_silo	90	Silo origen del que se dosifica
Tipo_materia	Prima	Tipo de materia origen
Tipo_destino	Premezcla	Tipo de materia destino
Densidad	1.0	Densidad de la materia origen
Tam_mezcla	500	Suma teórica total de los ingredientes al finalizar todas las dosificaciones
Desviacion	1.47	Error real en la dosificación

Como se puede observa, la información almacenada refleja el punto de partida de la dosificación, los elementos que intervienen, es tipo de materia a obtener y el origen de esta y finalmente los resultados obtenidos de esa dosificación.

Un dato que puede que sea bastante relevante, pero del que no disponemos, es la cola de caída utilizada, que viene fijada por las dosificaciones anteriores en unas circunstancias similares.

En los datos que se muestran en el ejemplo se describe una dosificación realizada en agosto de 2013, en la que se partía con la bascula vacía, se dosifica una materia prima con código 13, para obtener después de mezclar todas las materias dosificadas una materia con código 66. Teóricamente queremos obtener una cantidad de 500 Kg cuando tengamos todas las materias en la báscula y de esta concretamente queremos dosificar 235.026. Después de unos 50 segundos activando el elemento dosificador, se registra que la báscula ha variado de peso 236.5 Kg, por lo que se ha producido una desviación de 1,47 Kg sobre el objetivo.

En este caso, la desviación esta dentro de los limites de tolerancia y ha sido bastante aceptable, pero en ocasiones esa desviación puede superar esos limites suponiendo un deterioro de la calidad de la materia a producir. Estas desviaciones ocasionales puede que se produzcan debido a que en el cálculo de las colas de caída no se estén teniendo en cuenta todas las variables necesarias, esto es así para acelerar ese proceso de aprendizaje de colas y obtener buenos resultados desde el principio.

En las siguientes etapas de este trabajo intentaremos caracterizar esas desviaciones ocasionales con toda la información de la que se dispone de las dosificaciones y alguna más que podamos aportar de nuestra experiencia y así poder entrenar un modelo que sea capaz de predecirlas.

Capítulo 3

Análisis exploratorio

3.1 Introducción

En este capítulo realizaremos un análisis exploratorio de los datos: tipos de datos, cantidad de datos, correlación entre variables numéricas, variables cualitativas, etc. para poder diseñar un correcto preprocesado de datos en fases posteriores. Para ello nos serviremos de un cuaderno de JupiterLab y mediante las librería de pandas, matplotlib y seaborn iremos realizando el analis.

3.2 Análisis exploratorio

En primer lugar, cargaremos nuestros datos en un dataframe de pandas su manipulación y análisis:

```
df_dosificaciones = pd.read_csv('datos/datos.csv', \
                sep=',', na_values='\\N', index_col='codigo', \
                parse_dates=['fecha_inicio', 'fecha_fin'])

df_dosificaciones.info()
```

Contamos con 2.220.628 registros comprobando que efectivamente todas las columnas se han cargado correctamente y que los tipos de datos son los correctos. A continuación, eliminamos los registros con valores perdidos, que en este caso son 46.989 y los registros en los que la columna manual tiene valor 1, lo cual indica que el registro pertenece a una dosificación realizada manualmente, la cual puede generar ruido en nuestros entrenamientos de modelos:

```
df_dosificaciones = df_dosificaciones.dropna()

df_dosificaciones = df_dosificaciones[df_dosificaciones['manual'] == 0]
```

Filtrado de datos y detección de outliers.

En el siguiente paso estudiaremos la distribución de valores y los estadísticos de las variables numéricas que más caracteriza nuestro problema como son la cantidad solicitada y la dosificada, el peso inicial de la bascula, el tamaño de la mezcla y la desviación final de la pesada, los dato obtenidos se muestran en la Tabla 3.1. Distribución y estadísticos de las principales variables numéricas:

```
df_dosificaciones[['cantidad_solicitada','cantidad_dosificada', \
        'peso_inicial','tam_mezcla', 'desviacion']].describe()
```

Tabla 3.1. Distribución y estadísticos de las principales variables numéricas

	C. Solicitada	C. Dosificada	P. Inicial	T. mezcla	Desviación
Count	2.173597e+06	2.173597e+06	2.173597e+06	2.173597e+06	2.173597e+06
Mean	1.551270e+02	1.549170e+02	3.572072e+02	3.830281e+03	2.099743e-01
Std	4.110575e+02	4.106909e+02	8.677987e+02	7.734728e+02	1.728964e+01
Min	0.000000e+00	0.000000e+00	-6.400000e+01	4.000000e+02	-8.760000e+02
25%	4.604878e+00	4.699997e+00	0.000000e+00	4.000000e+03	-1.005216e-01
50%	1.195493e+01	1.180000e+01	2.200000e+01	4.000000e+03	-3.051758e-06
75%	3.564392e+01	3.476250e+01	7.323750e+01	4.020000e+03	9.999919e-02

Max	8.667933e+03	5.777000e+03	6.429000e+03	4.500000e+03	7.963000e+03

Para visualizar mejor los datos obtenidos generaremos una gráfica de cajas de las variables anteriores **Figura 3.1.** Distribución y estadísticos de las principales variables numéricas.

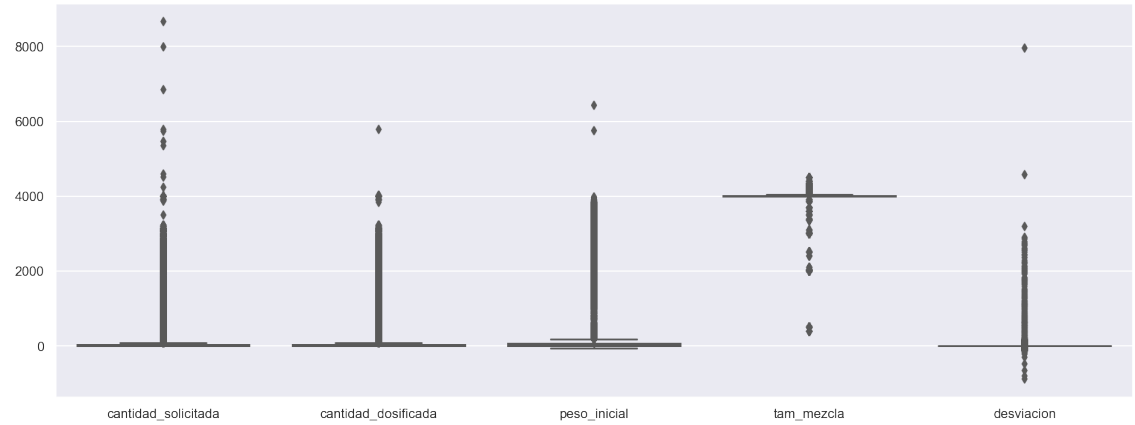

Figura 3.1. Distribución y estadísticos de las principales variables numéricas

A la vista de los datos podemos observamos que la cantidad solicitada normalmente no son superiores a unos 200Kg, esto sucede porque normalmente las formulas llevan uno dos productos base, de los que si se dosifica una gran cantidad, entre 2.000 o 3.000 Kg y posteriormente se van añadiendo cantidades más pequeñas de otros ingredientes. Cantidades solicitadas superiores a 2500 Kg las consideraremos outliers ya que no son habituales y normalmente puede que sean pruebas o errores.

Como era de esperar, los datos de la cantidad dosificada se asemejan bastante a los de la cantidad dosificada, ya que las desviaciones esporádicas pueden rondar los 100 Kg, dosificaciones mayores de 3000 Kg las desecharemos ya que también se pueden deber a pruebas o a errores mecánicos.

Los datos relativos al peso inicial denotan que normalmente la mayoría de las dosificaciones se producen cuando la bascula esta vacía o tiene poco peso, un poco acorde con las cantidades dosificadas, no son normales dosificaciones cuando la bascula

tiene ya un peso superior a 3500 Kg, por lo que desecharemos esos registros al considerarlos outliers.

En cuanto al tamaño de mezcla observamos que los datos son mas o menos normales, la mayoría de las mezclas se fabrica a 4000 Kg y las premezclas a 500 Kg, por lo que existen valores intermedios dentro de eses rango, pero son normales y correctos.

En cuanto a la desviación, que es la variable que queremos predecir, vemos que en general el sistema se comporta bien, ya que la media de la desviación es de un 0.2, y la desviación standard de 17 Kg, pero observamos que en ocasiones esa desviación es muy elevada. Como he comentado antes desviaciones superiores a 100 Kg suelen ser debidas a falta de materia en los silos o a errores mecánicos y estas circunstancias son difíciles de predecir, por lo que desviaciones superiores a esos 100 Kg las consideraremos outliers y las filtraremos.

Con la información obtenida y las decisiones tomas pasamos a filtrar los datos que consideramos ruido para nuestro aprendizaje:

```
df_dosificaciones =
df_dosificaciones[(df_dosificaciones['cantidad_solicitada'] > 0) &
                  (df_dosificaciones['cantidad_solicitada'] < 2500) &
                  (df_dosificaciones['cantidad_dosificada'] > 1) &
                  (df_dosificaciones['cantidad_dosificada'] < 3000) &
                  (df_dosificaciones['desviacion'] > -100) &
                  (df_dosificaciones['desviacion'] < 100) &
                  (df_dosificaciones['peso_inicial'] < 3500)]
```

Análisis de las columnas de información y creación de características.

Una vez filtrados los outliers analizaremos las columnas de las que disponemos y estudiaremos la posibilidad de crear nuevas características que nos puedan caracterizar el problema.

En primer lugar, debemos desechar las columnas 'fecha_fin' y 'cantidad_dosificada' ya que si las utilizásemos cometeríamos una fuga de datos, ya que es información que se genera una vez finalizada la dosificación.

Del mismo modo, la columna manual no nos aporta ninguna información, ya que siempre tiene el mismo valor, nos sirvió para eliminar los registros de las dosificaciones manuales, pero llegados a este punto no es de utilidad.

En cuanto a las variables 'densidad', 'tipo_materia' y 'tipo_destino', están estrechamente relacionadas con 'materia_origen' (ya tiene intrínseca la información de tipo_materia y densidad, las cuales no varían) y 'materia_destino' que siempre será del mismo tipo. Por tanto, prescindiremos de estas columnas ya que sería información redundante.

Por último, prescindiremos también de las columnas 'pesada' y 'mezcla' ya que cada vez que se cambia de pesada o mezcla las condiciones de las dosificaciones vuelven a ser las mismas, partiendo de la bascula vacía, por lo que no aportan gran información. A continuación, se expone el código de eliminación de dichas columnas:

```
columns_delete = ['fecha_fin','cantidad_dosificada','mezcla', \
                  'pesada','manual','densidad','tipo_materia', \
                  'tipo_destino']
df_dosificaciones.drop(columns=columns_delete, inplace=True)
```

En principio no tenemos ningún indicio de por que circunstancias pueden producirse las desviaciones, pero es fácil pensar que puede tener un carácter temporal, por lo que

crearemos una serie de características en este sentido, intentando aportar información al modelo. Para ello, descompondremos la fecha de la dosificación en diferentes columnas:

- Mes: mes en el que se produce la dosificación.
- Dia: día del mes en el que se produce la dosificación.
- Hora: hora del día a la que se produce la dosificación.
- Dow: dia de la semana a la que se produce la dosificación.

A continuación, se muestra el código para la creación de estas características:

```
df_dosificaciones['mes'] = pd.DatetimeIndex(df_dosificaciones['fecha_inicio']).month
df_dosificaciones['dia'] = pd.DatetimeIndex(df_dosificaciones['fecha_inicio']).day
df_dosificaciones['hora'] = pd.DatetimeIndex(df_dosificaciones['fecha_inicio']).hour
df_dosificaciones['dow'] =
pd.DatetimeIndex(df_dosificaciones['fecha_inicio']).dayofweek
df_dosificaciones.drop(columns=['fecha_inicio'], inplace=True)
```

Estudio de la correlación.

Una vez seleccionadas las características y filtrados los datos, estudiaremos grado de correlación entre variables, en la Tabla 3.2 Correlación entre variables y en la Figura 3.2 Correlación entre variables podemos observar los resultados.

Tabla 3.2 Correlación entre variables

	C.S	P.I.	M.O.	M.D.	SILO	T.M.	DES.	MES	DIA	HOR	DOW
C.S	1.000	0.262	0.289	0.005	-0.406	0.054	-0.018	-0.001	-0.000	0.010	0.005
P.I.	0.262	1.000	0.493	-0.072	-0.299	0.010	0.011	0.006	-0.001	-0.007	0.005
M.O	0.289	0.493	1.000	-0.015	-0.087	0.030	0.005	-0.021	-0.002	-0.006	0.001
M.D.	0.005	-0.072	-0.015	1.000	-0.004	-0.053	0.0003	-0.014	-0.005	0.050	-0.027
SILO	-0.406	-0.299	-0.087	-0.004	1.000	0.258	0.016	0.001	0.0002	-0.008	-0.003
T.M.	0.054	0.010	0.030	-0.053	0.258	1.000	0.011	0.018	-0.010	-0.044	0.011
DES.	-0.018	0.011	0.005	0.0003	0.016	0.011	1.000	-0.000	-0.000	-0.001	0.002
MES	-0.001	0.006	-0.021	-0.014	0.001	0.018	-0.000	1.000	0.012	0.057	0.000
DIA	0.000	-0.001	-0.002	-0.005	0.000	-0.010	-0.000	0.012	1.000	0.004	0.008

HOR	0.010	-0.007	-0.006	0.050	-0.008	-0.044	-0.001	0.057	0.004	1.000	0.041
DOW	0.005	0.005	0.001	-0.027	-0.003	0.011	0.002	0.000	0.008	0.041	1.000

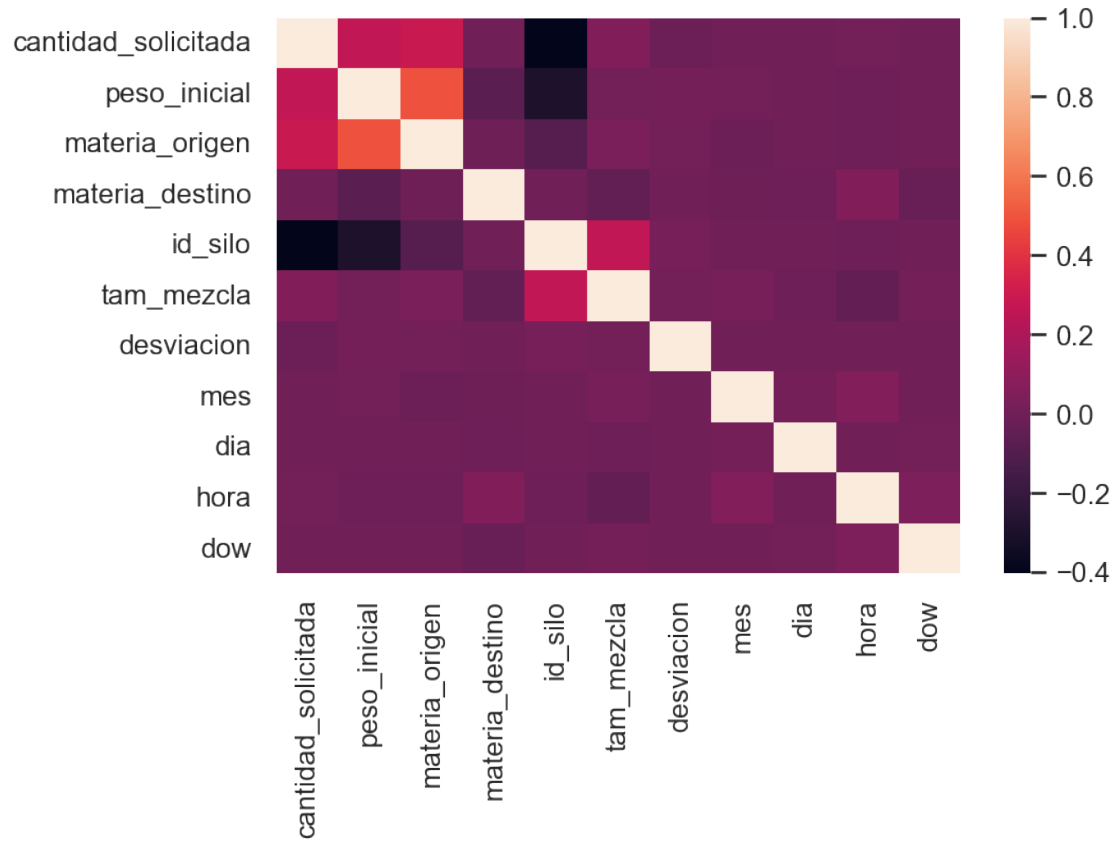

Figura 3.2 Correlación entre variables

Podemos observar que la mayor correlación se produce entre el peso inicial y la materia origen, eso es normal ya que como he comentado antes hay ciertos productos que sirven como base y se suelen dosificar al principio y el resto después y en orden ascendente de cantidad solicitada, por lo que normalmente los productos se dosifican con pesos iniciales similares en la bascula. Por la misma razón el peso inicial está correlacionado con la cantidad solicitada ya que las dosificaciones se producen por cantidad solicitada ascendentemente.

En cuanto a la variable a predecir que es la desviación, no vemos una correlación significativa con ninguna de las otras variables, quizá con la que mas tenga sea con la cantidad solicitada, esto es debido a que cuanto menor sea esa cantidad solicitada, menor probabilidad de desviación habrá.

Capítulo 4

Preprocesamiento

4.1 Introducción

En este capítulo abordaremos el preprocesamiento de los datos, mediante el cual realizaremos diferentes transformaciones de los datos para facilitar el aprendizaje de los algoritmos de machine learning. Esto es importante para evitar el sobreajuste de modelos debido a que ciertas características de los datos.

4.2 Preporcesamiento

En primer lugar, distinguiremos en tres las variables numéricas y categóricas, ya que recibirán un tratamiento diferente en este preprocesamiento.

- Variables Categóricas:
  - Materia_origen: Diferentes materias primas que pueden ser dosificadas, puede tomar un número finito de valores.
  - Materia_destino: Similar a la materia origen, son las diferentes formulas o recetas que queremos fabricas, también un número acotado de valores.
  - Id_silo: Silo del que se realiza la dosificación.
  - Mes, día, hora y día de la semana: También pueden tomar un número limitado de valores por lo que las consideraremos variables categóricas.

- Variables Numérica:
  - Cantidad_solicitada: Cantidad de materia objetivo a dosificar.
  - Peso_inicial: Peso actual de la bascula.
  - Tam_mezcla: Cantidad total teórica a dosificar sumando todos los productos.

Una vez identificados los tipos de las diferentes variables diseñaremos dos pipiline para el preprocesamiento de estos, para las variables numéricas programaremos dos tareas, una imputación de valores nulos, y un escalado standard de los valores. En cuanto a las variables categóricas también imputaremos los valores perdidos y realizaremos un onehotencoder para binarizar todas las variables categóricas:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

cat_features = ['materia_origen', 'materia_destino', 'id_silo', 'mes', 'dia', 'hora', 'dow']
num_features = ['cantidad_solicitada','peso_inicial','tam_mezcla']

num_transformer = Pipeline([('imputer',SimpleImputer()),
                    ('standard_scaler', StandardScaler())])

cat_transformer = Pipeline([('simple_imputer', SimpleImputer(strategy='constant')),
                    ('one_hot_encoder', OneHotEncoder(handle_unknown='ignore'))])

column_trans = ColumnTransformer(
            transformers=[
               ('num',num_transformer,num_features),
               ('cat',cat_transformer,cat_features)])
```

4.3 División train y test

Una vez diseñado el preprocesamiento haremos una división de nuestros datos en train y test, el conjunto de datos de train lo utilizaremos para entrenar nuestros modelos y el de test para validarlos. Dejaremos para los datos de test un 33% de los datos.

```
from sklearn.model_selection import train_test_split

X = df_dos_sample.drop('desviacion',1)
y = df_dos_sample['desviacion']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0)
```

Capítulo 5

Entrenamiento de Modelos

5.1 Introducción

Una vez analizados los datos, preprocesados y divididos en train y test, nuestra siguiente tarea será entrenar diferentes modelos de regresión para poder compararlos y quedarnos con el que mejores resultados obtengamos para pasarlo a producción. Para entrenar los diferentes modelos nos serviremos los modelos que nos proporciona la librería sklearn para Python.

5.2 Regresión Lineal

La regresión lineal es el modelo de regresión mas básico mediante el que se intenta representar la relación lineal entre cada una de las variables y la variable a predecir, a continuación, se muestra el código ejecutado para el entrenamiento, para la validación del modelo utilizaremos el error cuadrático medio que nos expresa claramente la desviación que se produce en las predicciones.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
X_train_prep = column_trans.fit_transform(X_train)
X_test_prep  = column_trans.transform(X_test)

linr_model = LinearRegression()

linr_model.fit(X_train_prep, y_train)
y_pred = linr_model.predict(X_train_prep)
y_pred_test = linr_model.predict(X_test_prep)

print("Error de entrenamiento: ", mean_squared_error(y_train, y_pred))
print("Error de test: ", mean_squared_error(y_test, y_pred_test))
```

```
Error de entrenamiento:  11.623373983850598
Error de test:  11.877618831629924
```

En términos generales vemos que el modelo se comporta bastante bien y el error no es muy elevado, por otro lado, también destacamos que no se ha producido un sobreajuste en el modelo y los resultados en el test también son satisfactorios y muy parecidos a los de entrenamiento.

5.3 Ridge

La regresión lineal con regularización intenta evitar el sobreajuste, en nuestro caso hemos visto que la regresión lineal simple ha funcionado correctamente y no se ha producido este fenómeno, pero quizá esa regularización nos ayude a filtrar o dar menos peso a las variables que menos caracterizan nuestro problema y así conseguir mejores resultados.

En este caso realizaremos una búsqueda de hiperparametros para el parámetro alpha y para la imputación de valores perdidos de los valores numéricos, para ello realizaremos una validación cruzada con una parte de los datos de test y posteriormente entrenaremos el modelo con los mejores parámetros obtenidos. Como método de validación utilizaremos nuevamente el error cuadrático medio.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

ridge_model = Ridge();

churn_pipe_lr = Pipeline([('prep', column_trans),
                          ('clas', ridge_model)])

parameters = {}
parameters['prep__num__imputer__strategy'] = ['mean', 'median']
parameters['clas__alpha'] = [1,0.1,0.01,0.001,0.0001,0]

X_sample = X_train.sample(n=200000)
y_sample = y_train[X_sample.index]

GS = GridSearchCV(estimator=churn_pipe_lr, param_grid=parameters,
scoring='neg_mean_squared_error', cv=5)
GS = GS.fit(X_sample, y_sample)

print("Mejor score: ", GS.best_score_)
print("Mejore configuración de parámetros: ", GS.best_params_)
```

```
Mejor score:  -12.201113251154016
Mejore configuración de parámetros:  {'clas__alpha': 1,
'prep__num__imputer__strategy': 'mean'}
```

A continuación, procedemos a entrenar el modelo Ridge con el mejor valor del parámetro alpha obtenido (1).

```
ridge_model = Ridge(alpha=1);

ridge_model.fit(X_train_prep, y_train)

y_pred = ridge_model.predict(X_train_prep)
y_pred_test = ridge_model.predict(X_test_prep)

print("Error de entrenamiento: ", mean_squared_error(y_train, y_pred))
print("Error de test: ", mean_squared_error(y_test, y_pred_test))
```

```
Error de entrenamiento:  11.623418665002474
```

Error de test: 11.877680102610482

Observamos que los resultados obtenidos son similares a la regresión lineal simple, seguramente debido a que al no haberse producido un sobreajuste en ese modelo no hemos conseguido mejorarlo.

5.4 Random Forest

Intentaremos mejorar los resultados obtenidos anteriormente entrenado un modelo mediante Random forest, explotando así la aleatoriedad de los datos y esperando la mejora comentada.

En este caso realizaremos una búsqueda de hiperparametros para el parámetro en numero de estimadores, máximo numero de atributos y máximo numero de niveles de profundidad.

```
from sklearn.model_selection import RandomizedSearchCV, RepeatedKFold

from sklearn.ensemble import RandomForestRegressor

# Se combinan los pasos de preprocesado y el modelo en un mismo pipeline.
pipe = Pipeline([('prep', column_trans),
                 ('modelo',  RandomForestRegressor())])

# Optimización de hiperparámetros
#
================================================================================
=======
# Espacio de búsqueda de cada hiperparámetro

param_distributions = {
   'modelo__n_estimators': [50, 100, 200, 1000],
   'modelo__max_features': ["auto", 3, 5, 7],
   'modelo__max_depth'   : [None, 3, 5, 10, 20]
}
```

```
# Búsqueda random grid
grid = RandomizedSearchCV(
       estimator  = pipe,
       param_distributions = param_distributions,
       n_iter     = 20,
       scoring    = 'neg_root_mean_squared_error',
       cv         = RepeatedKFold(n_splits = 5, n_repeats = 3),
       refit      = True,
       verbose    = 0,
       random_state = 123,
       return_train_score = True
      )

X_sample = X_train.sample(n=2000)
y_sample = y_train[X_sample.index]

grid.fit(X = X_sample, y = y_sample)

print("Mejor score: ", grid.best_score_)
print("Mejore configuración de parámetros: ", grid.best_params_)
```

```
Mejor score:  -3.424919299009838
Mejore configuración de parámetros:  {'modelo__n_estimators': 1000,
'modelo__max_features': 7, 'modelo__max_depth': 3}
```

A continuación, procedemos a entrenar el modelo Random Forest con el mejor valor del parámetro n_estimators (1000), max_features (7) y max_depth (3).

```
rf_model =
RandomForestRegressor(n_estimators=1000,max_features=7,max_depth=3)

rf_model.fit(X_train_prep, y_train)

y_pred = rf_model.predict(X_train_prep)
y_pred_test = rf_model.predict(X_test_prep)

print("Error de entrenamiento: ", mean_squared_error(y_train, y_pred))
print("Error de test: ", mean_squared_error(y_test, y_pred_test))
```

```
Error de entrenamiento:  11.734173725994342
```

Error de test: 11.996220816005833

Observamos que los resultados obtenidos son similares a los modelos anteriores, seguramente debido a que tiene una linealidad que es fácil representar con un modelo mas sencillo.

Capítulo 6

Comparación de modelos

6.1 Introducción

En este capítulo compararemos los resultados obtenidos para los diferentes modelos y decidiremos cual de ellos pasamos a producción para predecir las desviaciones en las dosificaciones.

6.2 Tabla comparativa de modelos

En la siguiente tabla **Tabla 6.1.** Comparación de datos de validación de modelos, visualizaremos los resultados de validación de los diferentes modelos tanto para los datos de entrenamiento como para test.

Tabla 6.1. Comparación de datos de validación de modelos

Modelos	Error Entrenamiento	Error Test
R. Lineal	11.62	11.87
Ridge	11.62	11.87
Random Forest	11.73	11.99

Capítulo 7

Conclusiones y Trabajo Futuro

7.1 Conclusiones

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

7.2 Trabajo futuro

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Bibliografía

Alexa Internet Inc. (2014). The top 500 sites on the web. Retrieved February 11, 2014, from http://www.alexa.com/topsites

Anda, B., Sjøberg, D., & Jørgensen, M. (2001). Quality and Understandability of Use Case Models. In 15th European Conference on Object-Oriented Programming (ECOOP'01) (pp. 402–428). London, UK: Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-45337-7_21

Babar, M. A., Kitchenham, B. A., Zhu, L., Gorton, I., & Jeffery, R. (2006). An empirical study of groupware support for distributed software architecture evaluation process. Journal of Systems and Software, 79(7), 912–925. https://doi.org/10.1016/j.jss.2005.06.043

Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). The Goal Question Metric Approach. In Encyclopedia of Software Engineering (Vol. 2, pp. 528–532). Wiley. Retrieved from http://wwwagse-old.informatik.uni-kl.de/pubs/repository/basili94b/encyclo.gqm.pdf

Basili, V. R., Shull, F., & Lanubile, F. (1999). Building Knowledge through Families of Experiments. IEEE Transactions on Software Engineering, 25(4), 456–473. https://doi.org/10.1023/A:1009742216007

Bereiter, C. (2002). Education and Mind in the Knowledge Age (1st ed.). Routledge. Retrieved from http://www.amazon.com/Education-Mind-Knowledge-Carl-Bereiter/dp/0805839437

Biostat Inc. (2006). Comprehensive Meta-Analysis. Retrieved April 17, 2013, from http://www.meta-analysis.com/

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). The Unified Modeling Language User Guide (2nd ed.). Addison-Wesley Professional.

Canfora, G., Cimitile, A., Garcia, F., Piattini, M., & Visaggio, C. A. (2006). Evaluating advantages of test driven development. In 2006 ACM/IEEE international symposium on International symposium on empirical software engineering (ISESE'06) (p. 364). Rio de Janeiro, Brazil: ACM Press. https://doi.org/10.1145/1159733.1159788

Castro, J., Kolp, M., & Mylopoulos, J. (2001). A requirements-driven development methodology. In 13th Int. Conf. On Advanced Information Systems Engineering (CAiSE'01) (pp. 108–123). London, UK: Springer-Verlag. https://doi.org/10.1007/3-540-45341-5_8

Celko, J., Davis, J. S., & Mitchell, J. (1983). A demonstration of three requirements language systems. ACM SIGPLAN Notices, 18(1), 9–14. https://doi.org/10.1145/948093.948094

Cockburn, A. (2000). Writing Effective Use Cases (1st ed.). Addison-Wesley Professional.

Cruz-Lemus, J. A., Genero, M., Caivano, D., Abrahão, S., Insfrán, E., & Carsí, J. A. (2011). Assessing the influence of stereotypes on the comprehension of UML sequence diagrams: A family of experiments. Information and Software Technology, 53(12), 1391–1403. https://doi.org/10.1016/j.infsof.2011.07.002

Cruz-Lemus, J. A., Genero, M., Manso, M. E., Morasca, S., & Piattini, M. (2009). Assessing the understandability of UML statechart diagrams with composite states—A family of empirical studies. Empirical Software Engineering, 14(6), 685–719. https://doi.org/10.1007/s10664-009-9106-z

Cruz-Lemus, J. A., Maes, A., Genero, M., Poels, G., & Piattini, M. (2010). The impact of structural complexity on the understandability of UML statechart diagrams. Information Sciences, 180(11), 2209–2220. https://doi.org/10.1016/j.ins.2010.01.026.

Cysneiros, L. M., & Yu, E. S.-K. (2004). Non-Functional Requirements Elicitation. In J. C. S. do Prado Leite & J. H. Doorn (Eds.), Perspectives on Software Requirements (pp. 115–138). Springer US. https://doi.org/10.1007/978-1-4615-0465-8_6

Damian, D. (2001). An empirical study of requirements engineering in distributed software projects: is distance negotiation more effective? In 8th Asia-Pacific Software Engineering Conference (APSEC'01) (pp. 149–152). Macao, China: IEEE Comput. Soc. https://doi.org/10.1109/APSEC.2001.991471

De Lucia, A., Fasano, F., Oliveto, R., & Tortora, G. (2006). Can Information Retrieval Techniques Effectively Support Traceability Link Recovery? In 14th IEEE International Conference on Program Comprehension (ICPC'06) (pp. 307–316). Athens, Greece: IEEE. https://doi.org/10.1109/ICPC.2006.15

Dieste, O., Fernández, E., García Martínez, R., & Juristo, N. (2011). Comparative analysis of meta-analysis methods: when to use which? In 15th International Conference on Evaluation & Assessment in Software Engineering (EASE'11) (pp. 36–45). Durham, UK: IET.

Dourish, P., & Bellotti, V. (1992). Awareness and coordination in shared workspaces. In ACM conference on Computer-supported cooperative work (CSCW'92) (pp. 107–114). Toronto, Canada: ACM Press. https://doi.org/10.1145/143457.143468