

O'REILLY®

# Machine Learning Design Patterns

Solutions to Common Challenges in Data Preparation, Model Building, and MLOps



Valliappa Lakshmanan,  
Sara Robinson & Michael Munn

Resumen de conceptos clave

Realizado por José Manuel Cuesta Ramírez

## 0. Introducción

Este libro describe una serie de problemas y retos definidos como patrones, que se encuentran normalmente en el desarrollo de proyectos de Machine Learning. Para cada uno de estos patrones, en el libro se detallan una amplia variedad de soluciones y los inconvenientes de cada una de ellas.

Este documento intenta cubrir algunos de los conceptos clave desarrollados en el libro a modo esquemático.

## 1. Necesidad de diseño de patrones en el Machine Learning

Los problemas de Machine Learning están sujetos siempre a retos como la calidad del dato ("Data Quality"), Reproducibilidad, Data Drift y Escalado de los servicios entre otros, por lo que debemos tener en cuenta criterios como:

- Si las fuentes empleadas provienen de sensores, es importante verificar que la calibración de todos es la misma, para no crear predicciones con modelos no precisos ni confiables.
- Los modelos de ML, tienen un elemento inherente random dado que los pesos son inicializados con valores aleatorios (empleo de semillas para evitar esto).
- La reproducibilidad de los modelos también debe tener en cuenta las dependencias de los frameworks empleados. Por ejemplo, si en una versión, en `train()` se realizan 13 llamadas a `rand()`, y en otra versión se realizan 14, emplear diferentes versiones causará ligeras diferencias en los resultados con los mismos datos y código.
- Mientras que los modelos de ML típicamente representan una relación estática entre los inputs y los outputs, los datos pueden cambiar significativamente a lo largo del tiempo (Data Drift). Para solucionar este problema, se necesita actualizar el conjunto de datos históricos y reentrenar los modelos para modificar los pesos de forma activa.

## 2. Estrategias para una representación óptima de los datos (Trucos para tratar los inputs y obtener mejores predicciones)

En este apartado se muestran consejos para el procesamiento de datos basados en el **escalado de las variables, el tratamiento de Outliers y el uso de variables categóricas**.

- A menudo como los algoritmos de ML usan un optimizador que está ajustado para trabajar con números en un rango  $[-1,1]$ , realizar un escalado de las variables a este rango puede ser beneficioso. El motivo, se debe a que los optimizadores del descenso por gradiente requieren más pasos para converger cuando la curvatura de la función de pérdidas se incrementa. Por tanto, en este rango, la función de error es más "esférica", y el coste computacional de entrenamiento suele ser menor, ya que los modelos convergen más rápido.

- Existen algoritmos que pueden ver afectado su rendimiento por la existencia de escalas muy diferentes en las variables: Por ejemplo, K-Means, si emplea distancia euclídea como medida de proximidad, confiará fuertemente en aquellas variables de mayor magnitud. Además, puede afectar negativamente a los algoritmos incluso si se emplean técnicas de regularización como L1 o L2. El empleo de este tipo de técnicas (L1-L2), “penaliza” la existencia de magnitudes grandes en los pesos de las ecuaciones, y puesto que estos pesos dependen a su vez de la magnitud de las variables, la regularización no afecta de la misma forma a todas las variables (existe menos control).

L1 Regularization

$$\text{Cost} = \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M |W_j|$$

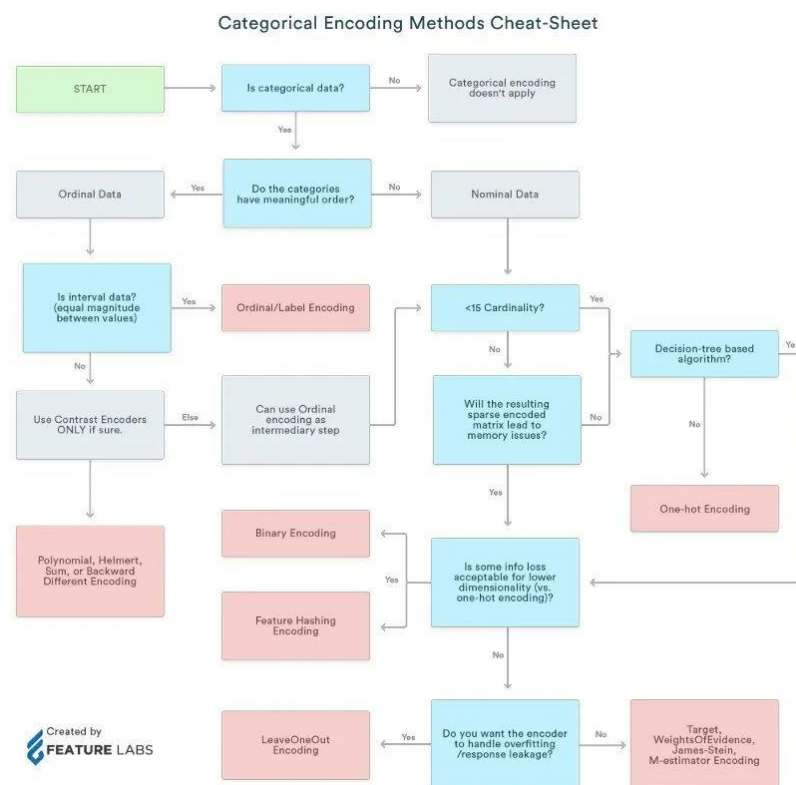
L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^M W_j^2}_{\text{Regularization Term}}$$

- Existen diferentes técnicas de escalado que pueden ser utilizadas :
  - Min-max scaling: Escalado lineal empleando el menor y mayor valor del conjunto de train. El problema de este enfoque es que estos extremos suelen ser valores atípicos.
  - Clipping: Ayuda a dirigir el problema de los Outliers empleando valores “razonables” en lugar de los extremos del conjunto de train.
  - Z-score normalization: Maneja los valores atípicos sin necesidad de un conocimiento previo del conjunto de datos empleando el valor medio y la desviación estándar del conjunto de train. (Esta es la mejor elección para distribuciones normales de datos).
  - Winsorizing: Esta técnica por sí sola no es una técnica de normalización, ya que se emplea para “limitar” los extremos de las variables. Por ejemplo, limitar los valores inferiores al percentil 1st a este valor, o limitar los valores superiores al percentil 99th a este valor. Tras aplicar esta técnica, se puede usar min-max scaling.
- En algunos problemas, la distribución de los valores de las variables no está uniformemente distribuida (curva de campana). En aquellos casos más extremos, es recomendable aplicar una transformación no lineal antes de aplicar un escalado. Un enfoque típico es el uso del logaritmo, aunque también es posible emplear raíces cuadradas, cúbicas etc.
- Box-Cox es una técnica paramétrica basada en logaritmos que toma un único parámetro (lambda) con el objetivo de controlar la “heterocedasticidad” y “sesgo” en la distribución de los errores así que la varianza no dependa de la magnitud.
- Pueden existir inputs definidos por arrays de números con una longitud variable. Las típicas técnicas para tratar este tipo de variables son: Creación de “feature” con la longitud de array,

“feature” con el valor medio, “features” con el mínimo y con el máximo o “features” con valores representativos de la distribución como los percentiles (10th, 20th...) entre otras.

- Existen ocasiones en las que puede ser útil transformar valores numéricos a valores categóricos, entre ellos: Cuando el input numérico es un índice (Por ejemplo el día de la semana, ya que en cada país empieza en un día diferente), Cuando la relación entre el input y la etiqueta no es continuo (creciente o decreciente), Cuando es ventajoso bucketizar las variables numéricas (Por ejemplo puede ser útil crear “bins” con la edad de los clientes , o crear una variable booleana para definir si es fin de semana o no).
- El enfoque más común al transformar las variables categóricas es el uso de One-Hot Encoding. A continuación, se muestra un Cheatsheet para la selección del método de Encoding según las características del problema (No pertenece a este libro):



- Para variables input que sean Arrays con valores categóricos:

Imaginemos que se quiere desarrollar un modelo de natalidad en función del tipo de nacimientos previos de la madre. Ejemplo de input: [Inducido, Inducido, Natural, Cesárea].

En este caso se puede realizar un conteo de las ocurrencias de cada tipo [2,1,1], u otro enfoque sería representar las frecuencias que suponen con respecto al total de nacimientos cada tipo, esto es, [0.5, 0.25, 0.25]. Finalmente se pueden generar 3 columnas que alberguen estos valores según la opción elegida.

### 3. Patrón Hashed Feature

**Este apartado cubre problemas asociados con variables categóricas en situaciones de vocabulario incompleto (NLP), variables con una cardinalidad muy alta o “cold start” (nuevos valores en la variable).**

Imaginemos un modelo para predecir los retrasos que se producirán en un vuelo en un dataset con 347 aeropuertos. Hay que tener en cuenta que además de los existentes, nuevos aeropuertos pueden ser contruidos o entrar en funcionamiento.

Para este tipo de situaciones, una solución es el uso de Hashed Features. Esta técnica genera grupos aleatorios asignando como “identificador” valores numéricos enteros en la misma escala para cada grupo. Una buena regla para elegir el número de hash buckets, es aquel que asigna para cada bucket en torno a 5 valores. Este enfoque a pesar de tener pérdidas de información puede hacer manejables las variables con una cardinalidad excesivamente alta.

La parte positiva de esta técnica será que en aquellas situaciones de missing values o de nuevos aeropuertos se podrán alcanzar unos resultados aceptables en base a los aeropuertos de su bucket.

En contraparte, empleando este tipo de técnica se pierde información y debemos estar dispuestos a que dos aeropuertos compartan el mismo grupo (en este problema de vuelos), ya que aunque realicemos un número muy alto de “buckets” siempre existe esta posibilidad.

Una posible “solución” para paliar las desventajas y obtener las ventajas de este enfoque, es realizar buckets de forma controlada en base a las características de los aeropuertos. Por ejemplo, crear grupos de 5-10 aeropuertos en función de la cantidad de años que llevan funcionando, o crear grupos en función de la probabilidad de realizar un vuelo “on-time” según los datos del conjunto de “train”.

### 4. Patrón Embeddings

**Este apartado cubre problemas relacionados con variables que poseen alta cardinalidad donde mantener la relación de cercanía entre las muestras de datos es esencial.**

Los Embeddings son representaciones de datos de alta cardinalidad en un espacio de menor dimensión de tal forma que la información relevante para el aprendizaje del problema es preservada.

One-hot Encoding es una forma común de representar variables categóricas, sin embargo, trata las variables de forma independiente. En algunos problemas mantener esta relación puede ser valiosa. Por ejemplo, en un problema de natalidad “mellizos” debería estar más cercano a “trillizos” que a “quintillizos”.

Esta capacidad de capturar las relaciones de similitud puede ser empleada para sustituir técnicas de clustering y de reducción de dimensionalidad como PCA.

Para seleccionar la dimensión más adecuada con el objetivo de perder la mínima información posible, existen varios enfoques recomendados. A continuación, se mencionan dos enfoques, y realizar un ajuste de hiperparámetros entre el rango definido por ambos, podría proporcionar en principio grandes resultados:



- Raíz cuarta del número total de elementos categóricos (únicos).
- 1.6 veces la raíz cuadrada de elementos únicos en la categoría.

En Tensorflow los Embedding se implementan de la siguiente manera:

```
tf.feature_column.embedding_column(columna_feature, dimension=2)
```

Hasta este punto, se ha explicado las ventajas de Embeddings en problemas con datasets tabulares, sin embargo, los Embeddings pueden ser empleados en texto como se explica a continuación e imágenes (con el empleo de Autoencoders, por ejemplo):

- **Text Embeddings (pag 42)**

Para implementar un “Text Embedding” en Keras, primero debemos crear una tokenización para cada palabra dentro del vocabulario español, inglés o en el que estemos trabajando. La tokenización es un “lookup table” que mapea cada palabra en el vocabulario con un índice (valor entero).

```
tokenizer = Tokenizer()
```

```
tokenizer.fit_on_texts(titles_df.title)
```

Una vez generado el diccionario de palabras, podemos invocar el mapping , y transformar los input de texto que sean transferidos a la instancia “tokenizer”, a un array de números enteros según el índice de cada una de las palabras.

```
Integerized_titles = tokenizer.texts_to_sequences(titles_df.title)
```

La instancia contiene otra información relevante que será usada después para la creación de la capa de Embedding. En particular VOCAB\_SIZE captura el número de elementos del del diccionario de palabras y MAX\_LEN contiene la longitud de la palabra máxima dentro del diccionario.

```
VOCAB_SIZE = len(tokenizer.index_word)
```

```
MAX_LEN = max(len(sequence) for sequence in Integerized_titles)
```

Puesto que no todos los textos de entrada tendrán la misma longitud, previo a la construcción del modelo es necesario aplicar un paso de padding para “homogeneizar” los tamaños de entrada:

```
From Tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
Def create_sequences(texts, max_len=MAX_LEN):
```

```
    Sequences = tokenizer.texts_to_sequences(texts)
```

```
    padded_sequences = pad_sequences(sequences,max_len, padding='post')
```

```
    return padded_sequences
```

A continuación, podemos construir un modelo DNN en Keras que implemente en primer lugar una capa de Embedding para transformar los números enteros de las palabras en “dense vectors” (pudiendo preservar las relaciones inherentes del texto).

Por otro lado, modelos preentrenados de text-embedding como Word2Vec, NNLM, GLoVE, o BERT pueden ser utilizados para procesar las ‘features’ de texto y que sean incluidas a datos tabulares para la construcción de modelos de Machine Learning con datos mixtos (tabulares y de texto).

En resumen, los Embeddings nos permiten extraer la similitud inherente entre dos categorías y puesto que nos da la representación numérica en un formato vector, podemos cuantificar de forma precisa la similitud entre estas categorías. El problema de esta técnica, es que cuando se aplica, se produce una pérdida de información, aunque se obtenga la relación o cercanía de los elementos, y por este motivo es muy importante analizar la “mínima dimensión” que es aconsejable utilizar .

## 5. Patrón Feature Cross

Este patrón hace referencia a la creación de variables como resultado de la combinación de dos o más columnas. Es una técnica típica en la fase de feature engineering y puede ayudar a los modelos a la interpretación de los datos.

Mientras que modelos más complejos como redes neuronales y árboles pueden aprender por sí mismos la relación entre variables (columnas del dataset), este tipo de técnicas permiten a modelos más simplistas como las regresiones lineales descifrar las relaciones internas del dataset.

**La forma de aplicarlo sobre dos columnas categóricas, es crear una variable, resultado de la combinación de casuísticas (combinaciones) entre las columnas originales. Para variables numéricas continuas, debemos aplicar inicialmente una ‘bucketización’ creando grupos (valores categóricos) para posteriormente cruzar las variables.**

Debido a que estas combinaciones pueden generar una variable con una cardinalidad muy alta (categórica), puede ser útil pasar la variable de ‘feature cross’ a través de una capa de Embedding para crear una representación con menor dimensión.

Por ejemplo, si trabajásemos con dos variables numéricas como latitud y longitud, en primer lugar, tendríamos que generar para cada variable grupos (agrupando por su valor en rangos). Tras esto, podríamos realizar el cruce de ambas variables, sin embargo, la cardinalidad podría ser muy alta si los grupos se generaron con rangos muy pequeños. En estos casos, para evitar su traducción mediante One-hot Encoding, es aconsejable emplear una capa de Embedding que además de realizar una reducción de dimensionalidad permita obtener la relación entre grupos.

Debemos ser conscientes del número de muestras que dispondremos para cada nuevo grupo generado en la variable de “feature cross”, ya que si es reducido puede llevarnos a overfitting y será necesario el uso de técnicas de regularización (L1 o L2).

También debemos de tener cuidado de que las variables que se someten a este proceso no estén altamente correlacionadas ya que el resultado es un producto cartesiano y por tanto no traería ningún valor.

## 6. Patrón Reframing

Este patrón o técnica se refiere a la transformación de los datos para pasar de un problema de regresión a un problema de clasificación o viceversa en función de las características y del interés.

Por ejemplo, si nos enfrentamos a un problema de predicción de lluvia en una región, intentar predecir la cantidad exacta, aumenta considerablemente la dificultad del problema y esto puede hacer que el modelo de regresión planteado, obtenga unos resultados que no son los esperados.

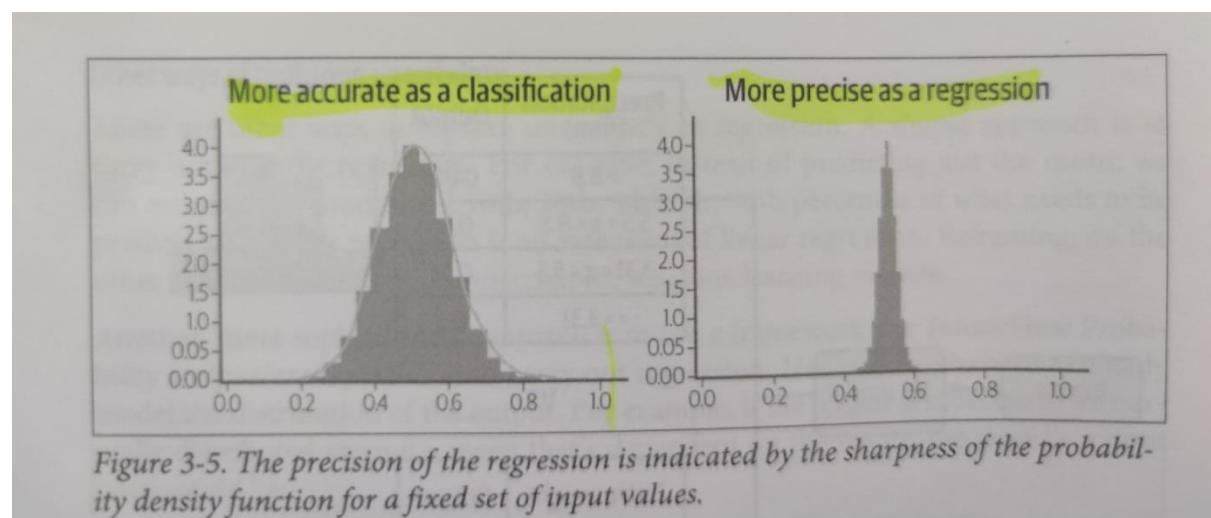
Para este tipo de situaciones, se puede discretizar en grupos la variable target, y realizar un modelo de multi-clasificación en el que obtengamos la probabilidad para cada una de las categorías. Esta simplificación del problema, nos hace perder algo de detalle en las predicciones (no conocemos la cantidad exacta), pero normalmente se mejora el rendimiento teniendo en cuenta un rango de precipitación. Además, en esta reestructuración, estamos consiguiendo limitar los valores extremos que el modelo predice, ya que el output es una categoría (Evitamos por ejemplo predecir valores inferiores a cero en problemas en los que no es viable un valor negativo).

En otras circunstancias, puede interesarnos traducir un problema de clasificación a un problema de regresión. Por ejemplo, en un sistema de recomendación de películas puede interesar pasar de un modelo que intenta predecir la próxima película recomendada (variable categórica) a un problema de regresión, en el que predecimos las características del usuario y una vez con este resultado, analizamos todas las películas cercanas a este “punto”.

Un enfoque interesante tanto para reestructurar como un problema más simple de regresión (regresión cuantil) como para transformar a un problema categórico, es el uso de los valores de los cuantiles (10th, 20th .. 90th).

- **Precisión de los modelos**

Aunque cada problema y modelo debe ser examinado de forma individual, en general dependiendo de lo “afilada” que sea la distribución de datos podemos indicar lo siguiente:





## 7. Patrón Multilabel

Esta problemática se refiere a situaciones en las que podemos asignar más de una etiqueta a un registro o muestra del conjunto de training. **Esto quiere decir que puede que nos enfrentemos ante un problema de clasificación de texto (por ejemplo) y que para una frase se deban activar varias categorías como output:**

La frase: “¿Cómo puedo dibujar un dataframe de pandas?”, podría activar las etiquetas de salida “Python”, “pandas” y “visualización” por ejemplo.

Este tipo de modelos también aplica a datos tabulares, por ejemplo, podemos tener un dataset de características de pacientes y que presenten más de una enfermedad o patología.

Por lo general los modelos predictivos buscan seleccionar una única categoría, para estos casos es común el uso de la función de activación “softmax” para la capa de salida (en redes neuronales) de tal manera que obtenemos un array de números cuya suma es igual a 1. Cada valor indica la probabilidad de que la muestra pertenezca a cada una de las categorías posibles. Por tanto, empleando “argmax” seleccionamos el índice de la mayor probabilidad del array, y con este la categoría seleccionada.

Para conseguir que los modelos tengan varios output como salida, la solución es usar una función de activación sigmoidea en la última capa, de tal manera que para cada categoría genere un valor entre 0 y 1. Cuando se supere el umbral seleccionado (threshold), la categoría será incluida en el output. Para su implementación solo es necesario incluir dicha función de activación en la capa “dense” de salida con tantas unidades como categorías tenga el problema.

Los problemas de clasificación binaria son los únicos que permiten el uso de la función sigmoidea en la capa de salida y puesto que los modelos con un solo nodo o unidad en el “output” son más eficientes éste es el enfoque más recomendado.

Otra técnica muy usada en problemas de clasificación multi-etiqueta es entrenar varios clasificadores binarios, este enfoque es conocido como “one versus rest”. Entre las ventajas, facilita la tarea predictiva, permite usar otras arquitecturas binarias como SVMs, y técnicas de rebalanceo, aunque tiene la desventaja de que aumentando el número de modelos se complica el mantenimiento del sistema predictivo.

## 8. Patrón Ensembles

Este patrón se refiere a la técnica del machine Learning que combina múltiples modelos predictivos y agrega sus resultados para generar unas predicciones definitivas. Los métodos más conocidos son: Bagging, Boosting y Stacking. Los modelos Ensemble pueden mejorar el rendimiento y los resultados de las predicciones de modelos individuales.

- **Bagging:** Para manejar alta varianza en los modelos de machine Learning. Se construyen “K” submodelos con “K” datasets separados que se generan aleatoriamente y donde se permite la repetición de valores (registros). El resultado del modelo es la media de todos los modelos en un problema de regresión, y la opción más votada en un problema de clasificación.

- **Boosting:** **Para manejar alto bias.** Combinación de modelos de forma secuencial e iterativa, de tal manera que, en cada iteración, el modelo se centre en corregir los residuos o errores de la iteración anterior.
- **Stacking:** Es un método de Ensemble muy amplio que puede ser usado con varios enfoques. Por ejemplo, podríamos usar una combinación de modelos con un enfoque similar a Bagging (valor medio de las predicciones de los modelos individuales en un problema de regresión) pero empleando diferentes tipos de modelos. También sería posible generar una primera capa de modelos cuyas predicciones sean usadas como “inputs” de una segunda capa de modelos que entregaría las predicciones definitivas.

## 9. Patrón Rebalancing

Este apartado proporciona varios enfoques para manejar datasets que se encuentran desbalanceados por su propia naturaleza. Hay que tener en cuenta que los modelos de Machine Learning aprenden mejor cuando se les proporciona un número similar de muestras para cada etiqueta en el dataset.

Esta situación aplica a muchos tipos de modelos, incluyendo clasificación binaria, clasificación multiclase, clasificación multietiqueta y regresión. En los casos de regresión, los datasets desbalanceados se refieren a datos con ‘Outliers’ que son o mucho mayores o menores a la media del dataset.

En primer lugar, dado que el “Accuracy” del modelo puede conducir a errores de interpretación en el rendimiento, es importante elegir una métrica apropiada cuando se desarrollan este tipo de modelos. Es mejor usar métricas como ‘Precision’, ‘Recall’ o ‘F-measure’ para conseguir una imagen completa de como nuestro modelo está actuando.

La ‘Precision’ indica de aquellos positivos que predice el modelo que porcentaje realmente lo son, el ‘Recall’ por su parte, el porcentaje de positivos que es capaz de capturar el modelo. Sin embargo, aunque estas métricas son muy útiles para evaluar como el modelo se comporta con respecto a la clase “positiva”, existen otras métricas como el área bajo la curva (AUC) que sirven como métrica para evaluar el comportamiento del modelo frente a las dos clases.

Las principales técnicas son: ‘Downsampling’, ‘Weighted Classes’ y ‘Upsampling’.

### Downsampling

Es una solución para manejar datasets desbalanceados que consiste en reducir el número de muestras de la clase mayoritaria para la fase de entrenamiento del modelo, mientras que se mantiene el número de muestras de la clase minoritaria. Este enfoque, aunque es más típico en problemas de clasificación, puede ser utilizado en problemas de regresión.

Cuando se realiza este enfoque es importante tener en cuenta que, al eliminar muestras, estamos perdiendo información almacenada que puede más o menos valioso y por tanto puede afectar a la habilidad del modelo para identificar la clase mayoritaria. Dicho esto, en términos generales incluso perdiendo parte de información su aplicación presenta mas beneficios que inconvenientes.

## Weighted Classes

Otro enfoque es cambiar el peso que nuestro modelo da a las muestras de cada clase. De esta manera, se puede controlar que ciertas etiquetas o clases reciban mayor importancia durante el entrenamiento.

En concreto, estos pesos que se entregan a cada clase, actúan en la función de pérdidas multiplicando al “error” y, por tanto, si el peso es mayor, el “error” generado por la función de pérdidas también lo será, y el modelo tiende a reducir el error en esta clase. Este enfoque, aunque es muy potente, puede no funcionar muy bien cuando trabajamos con un entrenamiento en ‘Batch’ (por ejemplo, en redes neuronales), ya que, si en ese ‘Batch’ no se recibe ninguna muestra de la clase minoritaria, aunque hayamos cambiado los pesos de penalización no se verá penalizada ninguna muestra.

En Keras, podemos pasar el parámetro ‘class\_weight’ a nuestro modelo cuando lo entrenemos (fit). Este parámetro es un diccionario que mapea cada clase con el peso que se le debería de asignar. Los pesos deben estar relacionados con la relación del volumen de muestras dentro del dataset.

En conjunto con asignar ‘class weights’ es útil inicializar los modelos en la capa de salida con un ‘bias’ para tener en cuenta el desbalanceo del dataset (Por defecto, keras usa bias = 0). Realizar esto, ayudará al modelo a converger más rápido, ya que el bias de la salida en media debería ser el logaritmo del ratio de la clase minoritaria a la clase mayoritaria. Por tanto, si realizamos esto al inicializar el modelo, no tendrá que ser descubierto a través del descenso por gradiente.

$$\text{Bias} = \log(\text{num\_minority\_examples}/\text{num\_majority\_examples})$$

## Upsampling

Otra técnica común es ‘upsampling’ con la que sobre representamos la clase minoritaria tanto duplicando las muestras de esta clase como generando muestras sintéticas. Esta técnica es usada normalmente en conjunto con ‘downsampling’ de la clase mayoritaria.

Un algoritmo usado para este enfoque es SMOTE que construye muestras sintéticas analizando el espacio de características de la clase minoritaria en el dataset y entonces genera muestras similares dentro de este espacio de características usando un enfoque de vecinos cercanos KNN.

El problema de este enfoque puede ocurrir cuando la creación de muestras se realiza en la “frontera” de separación entre las clases. En esta situación podría afectar negativamente a la detección del modelo.

También existen otros enfoques y técnicas que pueden generar grandes resultados como la reestructuración de un problema de regresión en un problema de clasificación con dos categorías: la clase mayoritaria con un volumen amplio del dataset y una clase minoritaria con los valores atípicos que nos gustaría detectar. Una vez realizada esta transformación, podríamos aplicar algunas de las técnicas anteriores. Otra técnica podría ser la creación de modelos ensemble con diferentes ‘subsets’ (con una proporción similar entre mayoritaria y minoritaria) con registros diferentes para la clase minoritaria en los ‘subsets’.

## 10. Patrón Checkpoints

Antes de comenzar con el desarrollo de los 'Checkpoints' propios de la fase de entrenamiento de un modelo de Deep Learning, es importante remarcar que para que un modelo pueda llegar a obtener buenos resultados, es necesario que sea capaz de sobreajustar sin aplicar técnicas de regularización. Si no lo consigue, entonces tendríamos que probar con una red neuronal más grande.

En los 'Checkpoints' se van almacenando de forma periódica y en base a unos criterios, el estado completo de los modelos parciales que se han ido obteniendo en un proceso de entrenamiento. Estos modelos parciales, pueden ser empleados como modelo definitivo en un 'early stopping' o como comienzo para continuar el entrenamiento en casos en los que haya fallado el computador.

Al final de cada época podemos guardar el estado, de tal manera que si el entrenamiento se ve interrumpido es posible continuar desde este modelo previamente guardado (Checkpoint). Es importante diferenciar que los modelos finales que son "guardados" o "exportados" sin este tipo de técnica no contienen toda la información (funciones de activación, pesos, Learning rate, etc) y solo disponen de la función de predicción necesaria para crear futuras predicciones.

### Early stopping

Si en un proceso de entrenamiento empezamos a sobreentrenar con el dataset de train, el error en validación comenzará cada vez a ser mayor y por tanto podemos detener el entrenamiento antes de lo esperado cuando sucedan más de 'N' Checkpoints sin mejora.

### Checkpoint selection

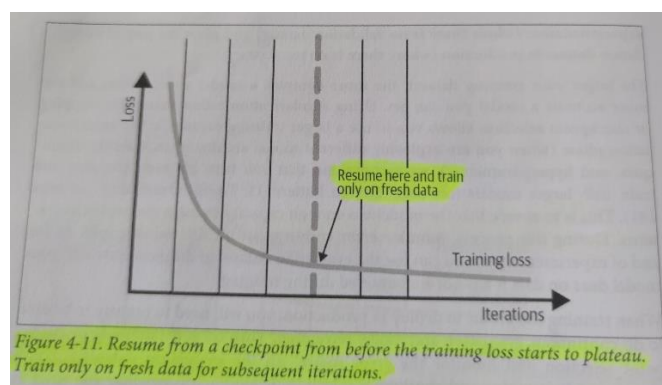
No se detiene el entrenamiento con un early stopping, se prolonga, y al finalizar, se selecciona el Checkpoint de la ejecución óptima, y a partir de este, se exporta dicho modelo.

### Regularization o Dropout

También pueden ser útiles el uso de regularizaciones como L1 o L2, u otras técnicas como 'Dropout' para evitar el sobre entrenamiento o el uso de early stopping por el riesgo de quedarnos con una solución subóptima.

### Fine-tuning

Para reentrenar el modelo con datos nuevos, y enfatizando en estos, la mejor opción es seleccionar un modelo previo a la estabilización y reentrenar únicamente con el nuevo dataset. Este enfoque evita reentrenar de nuevo modelos que son muy pesados y además se consigue que el modelo se comporte de forma óptima en las nuevas tendencias ocultas en los datos. (Es un proceso de Transfer Learning).



## Redefining an epoch

En ocasiones se observa como a la hora de entrenar un modelo de Deep Learning se utilizan en los ejemplos conjuntos de datos definidos como 'X\_train' e 'y\_train', sin embargo, este enfoque supone que el ordenador tiene la capacidad de cargar los datos en memoria e iterar el número de épocas definidas sin caer en un error de memoria. Para hacer este código más estable, es aconsejable proporcionar al "fit" un dataset de tipo Tensorflow y no un Numpy array. Esto proporciona capacidad de iteración y una carga de los datos perezosa.

Sin embargo, usar épocas en grandes datasets sigue siendo una mala idea por varios motivos:

- La diferencia de tiempo de entrenar un dataset 14.3 veces y 15 puede ser horas.
- Esperar un millón de muestras hasta el próximo Checkpoint puede ser demasiado tiempo.
- Los datasets crecen en el tiempo y por tanto, el problema será cada vez mayor.
- En enfoques de entrenamiento distribuido el concepto de época no está claro.

## Steps per epoch

En lugar de entrenar 15 épocas podemos decidir entrenar durante 143.000 'steps' donde el 'Batch size' sea 100 (14.3 épocas). De esta manera podemos controlar el número de 'Checkpoints' que queremos obtener del modelo, y aumentar o reducir el tamaño de la época. En cada 'step' se realiza la actualización de pesos. Este enfoque a secas, también puede dar problemas con la incorporación de nuevos datos, ya que el modelo podría analizar (observar) menos veces cada una de las muestras, dando lugar a problemas en la convergencia. Por tanto, la solución es el uso de 'Virtual epochs'.

## Virtual epochs

En este enfoque lo que se mantiene constante no es el número de 'steps' si no el número de veces que las muestras (registros) son vistas por el modelo (independientemente de la longitud del dataset).

```
NUM_TRAINING_EXAMPLES = 1000 * 1000
STOP_POINT = 14.3
TOTAL_TRAINING_EXAMPLES = int(STOP_POINT * NUM_TRAINING_EXAMPLES)
BATCH_SIZE = 100
NUM_CHECKPOINTS = 15
steps_per_epoch = (TOTAL_TRAINING_EXAMPLES //
                   (BATCH_SIZE * NUM_CHECKPOINTS))
cp_callback = tf.keras.callbacks.ModelCheckpoint(...)
history = model.fit(trains,
                    validation_data=evalds,
                    epochs=NUM_CHECKPOINTS,
                    steps_per_epoch=steps_per_epoch,
```

## 11. Patrón Transfer Learning

Antes de comenzar, debemos indicar que esta técnica tan conocida es principalmente usada en los problemas de imagen y texto, aunque existen ciertos modelos que de forma puntual pueden ser usados para datos tabulares (TabNet).

Previo a que un modelo aprenda los patrones definitivos lo primero que debe de analizar son los pixeles, ejes y formas que poseen las imágenes (En modelos de Deep Learning aplicados a imágenes).

Las primeras capas de estos modelos son destinadas al reconocimiento de dichas características básicas, mientras que las capas finales se dedican a clasificar por categorías o clases en base a la extracción realizada.

Por tanto, el objetivo es reusar las primeras capas de modelos complejos que han sido entrenados en tareas “similares” durante largos períodos de tiempo, dejar congeladas las capas y el valor de los pesos, y añadir nuevas capas que serán las que se sometan a entrenamiento para ajustar el modelo a nuestra tarea.

Algunos detalles que deben ser conocidos y considerados son:

- La última capa que permanece de estos modelos reutilizados se denomina Bottleneck.
- La capa “top” del modelo que podemos decidir si seleccionar o no, normalmente es el conjunto de capas a partir de la capa “flatten” incluida.
- Las imágenes incluidas pueden ser del mismo tamaño que indica la red preentrenada o también pueden tener un tamaño menor.
- Si queremos editar las capas de extracción de características, se pueden descongelar algunas capas convolucionales y permitir su reentrenamiento (fine-tuning).
- Si además el dataset es muy pequeño, es mejor no entrenar capas convolucionales ya que la red que estamos empleando (preentrenada) es muy compleja y podría caer fácilmente en overfitting.

## 11. Patrón Distribution Strategy

En este patrón se explica como el bucle de entrenamiento se lleva a cabo sobre múltiples Workers normalmente con “caching”, aceleración de hardware y paralelización. El objetivo es conocer como acelerar el entrenamiento de grandes redes neuronales.

Una forma de acelerar estos entrenamientos es mediante estrategias distribuidas en el bucle de entrenamiento. Hay diferentes formas, pero la idea común es dividir el esfuerzo entre múltiples máquinas. Existen dos estrategias a grandes rasgos en las que puede ser realizado: “data parallelism” y “model parallelism”.

En Data parallelism la computación es dividida entre diferentes máquinas y dentro de estas, entre diferentes Workers para entrenar diferentes subconjuntos de train. En Model parallelism el modelo es dividido y diferentes Workers llevan a cabo la computación para diferentes partes del modelo.

En esta sección nos centraremos principalmente en **Data parallelism con Tensorflow usando `tf.distribute.Strategy`**.



Para implementar Data parallelism debe existir un método localizado en cada uno de los Workers para computar el gradiente y compartir la información para hacer las actualizaciones del modelo. Podemos distinguir dos tipos de entrenamiento: Síncrono y asíncrono.

### Entrenamiento síncrono

En resumen: Para la actualización se realiza una media del resultado de cada GPU y no se avanza hasta que se obtenga el resultado de todos los Workers.


Cada Worker (normalmente una GPU) posee una copia del modelo y mediante un descenso por gradiente actúa sobre un mini-batch de datos que recibe. Existe un servidor central que sostiene el modelo más actual y realiza la actualización de pesos de acuerdo a los gradientes recibidos desde cada uno de los Workers (Por ejemplo, promediados). Una vez que los parámetros son actualizados, el nuevo modelo es enviado a los Workers con otro Split de datos (mini-batch) y el proceso se repite.

En TensorFlow, `tf.distribute.MirroredStrategy` soporta distribución síncrona entre múltiples GPU sobre la misma máquina. Para realizarlo en Keras es necesario crear una instancia previamente:

```
mirrored_strategy = tf.distribute.MirroredStrategy()
with mirrored_strategy.scope():
    model = tf.keras.Sequential([tf.keras.layers.Dense(32, input_shape=(5,)),
                                tf.keras.layers.Dense(16, activation='relu'),
                                tf.keras.layers.Dense(1)])
    model.compile(loss='mse', optimizer='sgd')
```

Existen otras distribuciones síncronas dentro de Keras como `CentralStorageStrategy` y `MultiWorkerStrategy`. `MultiWorker` habilita la distribución para ser extendida no solo entre GPUs de la misma máquina, si no, también entre múltiples máquinas. En `CentralStorage` las variables del modelo no son reflejas (en una GPU central), en lugar de esto, están localizadas en la CPU y las operaciones son replicadas entre todas las GPUs.

Para la decisión de la estrategia a elegir podemos usar el siguiente esquema:

 Table 4-2. Choosing between distribution strategies depends on your computer topology and how fast the CPUs and GPUs can communicate with one another

	Faster CPU-GPU connection	Faster GPU-GPU connection
One machine with multiple GPUs	<code>CentralStorageStrategy</code>	<code>MirroredStrategy</code>
Multiple machines with multiple GPUs	<code>MultiWorkerMirroredStrategy</code>	<code>MultiWorkerMirroredStrategy</code>

### Entrenamiento asíncrono

En resumen: No existe una fase “all-reduce” (No espera a que finalicen todos los Workers).

Los Workers entrenan sobre diferentes ‘slices’ del input data independientemente y se va actualizando el modelo conforme se van recibiendo el feedback de los Workers. El problema de este enfoque es que algunos ‘mini-batch’ poder ser perdidos durante el entrenamiento y es más difícil de tener un seguimiento de cuántas épocas realmente se han procesado.

En Keras ParameterServerStrategy, implementa una distribución asíncrona sobre múltiples máquinas, aunque también existen otras como OneDeviceStrategy (una máquina).

La elección entre el tipo de entrenamiento dependerá de la fortaleza que tengan las conexiones con la red y dependiendo de si todos los dispositivos recaen sobre el mismo 'host'.

Por último, hablaremos brevemente sobre Model parallelism:

En algunos casos, las redes neuronales son muy grandes y no pueden ser entrenadas en memoria en un solo dispositivo, por ejemplo, Google's Neural Machine Translation tiene miles de millones de parámetros. Para poder entrenar este tipo de modelos es necesario dividirlo entre varios dispositivos. Cada dispositivo opera sobre el mismo mini-batch, pero llevando a cabo las computaciones relacionadas con una parte específica del modelo.

## 12. Patrón Hyperparameter Tuning

El entrenamiento del modelo es insertado en un método de optimización para encontrar el conjunto de parámetros que alcanza los mejores resultados.

El método más conocido y típico es Grid Search, sin embargo, el problema de este enfoque es que no existe ninguna lógica que es aplicada a la hora de seleccionar diferentes combinaciones. Es una solución bruta donde se prueban todas las combinaciones sin tener en cuenta los resultados que va arrojando el modelo.

Una alternativa es RandomizedSearchCV, que ejecuta más rápido al no probar todas las combinaciones, pero permite analizar un espacio de parámetros más amplio al probar combinaciones aleatorias. Sin embargo, sigue teniendo el mismo problema de no aprender de los resultados.

La solución puede ser encontrada empleando la librería **"Keras-tuner" que realiza una optimización con una búsqueda Bayesiana**. Este enfoque aprende a encontrar la solución óptima dentro de un rango posible de valores y, básicamente, si observa que siguiendo una dirección de patrones en los parámetros los resultados van mejorando continúa en esa dirección. Teóricamente en vez de realizar nuestro entrenamiento completo en cada una de las combinaciones, se define una función que emula al modelo, pero es mucho más rápida de ejecutar que se encarga de **seleccionar combinaciones potenciales y solo se someterán al entrenamiento completo dichas combinaciones**. Los enfoques comunes al crear esta función son "Gaussian Process" o un "Tree-structured Parzen estimator".

Para su ejecución principalmente distinguimos 'hp.Int' (Enfoque bayesiano) y 'hp.Choice' (Más parecido al Grid Search):

```
# Tune the number of units in the first Dense layer
# Choose an optimal value between 32-512
hp_units = hp.Int('units', min_value=32, max_value=512, step=32)
model.add(keras.layers.Dense(units=hp_units, activation='relu'))
# Tune the learning rate for the optimizer
# Choose an optimal value from 0.01, 0.001, or 0.0001
hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
```

Los **algoritmos genéticos** son una alternativa al enfoque Bayesiano para la búsqueda de hiperparámetros, pero tienden a requerir mucho más entrenamiento y ejecuciones. Este enfoque, comienza definiendo una “fitness function” que mide la calidad de una combinación de parámetros. Tras ello, se toman un conjunto de combinaciones random que se evalúan con esta función y “la combinación ganadora” será tomada en cuenta para crear el siguiente espacio de búsqueda de parámetros. De forma iterativa, este proceso va conduciendo hacia el espacio óptimo, haciéndolo cada vez más estrecho.

## 13. Patrón Resilient Serving

Este apartado engloba conceptos teóricos desde el Patrón 16 al Patrón 20 del libro.

### Stateless Serving Function

El objetivo es abordar la situación en la que se reciben millones de peticiones por segundo para obtener las predicciones. Hay varios problemas llevando a cabo inferencias utilizando `model.predict()` en memoria, entre los que se destaca la necesidad de cargar el modelo entero de Keras en memoria, con todas sus capas o Embeddings y que el método tiene que ser enviado uno a uno.

La solución es exportar el modelo en un formato que capture el ‘Core’ de la fórmula matemática, restaurar dicho ‘Core’ como una función “Stateless” (una función cuyo output solo depende de los input que recibe) y por último desplegar en un framework que proporciona un REST Endpoint.

Para generar esa función emplearemos `saved_model_cli` (pag 206):

```
saved_model_cli show --dir ${export_path} \  
--tag_set serve --signature_def serving_default
```

### Continued Model Evaluation

Es importante detectar en que momento los modelos requieren de un proceso de re-entrenamiento.

Para ello, es necesario estar constantemente guardando las predicciones que fueron generadas por los modelos y evaluando su rendimiento una vez que dispongamos de las etiquetas reales. Cuando el modelo no alcance cierto umbral definido, le realizaremos el proceso de re-entrenamiento.

En conjuntos de datos en los que exista una cierta tendencia “creciente” o “decreciente” en el tiempo (Data Drift), será más difícil la generalización de los datos. En estos casos, algunas herramientas como TFX Data Validation pueden ser útiles para detectar anomalías, training-serving skew o data drift en los nuevos datos adquiridos.

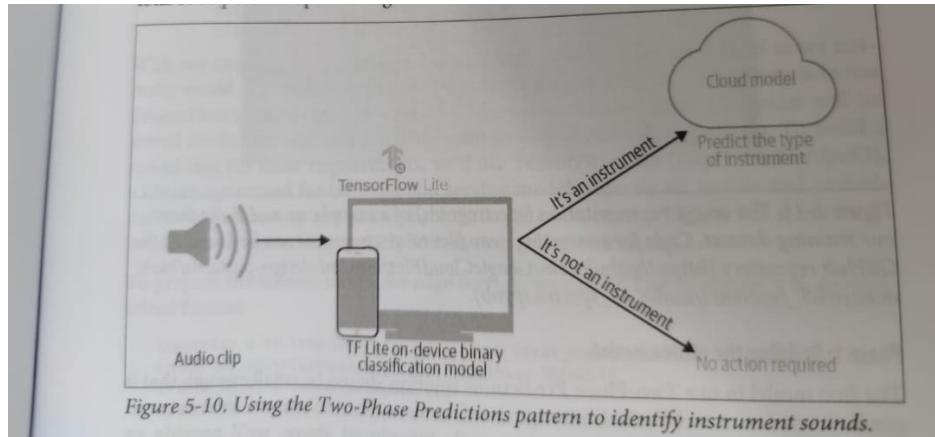
### Two-Phase Predictions

Cuando se despliegan modelos de machine Learning no podemos confiar siempre en que los usuarios finales dispondrán de unas conexiones perfectas de internet. Los modelos desplegados en el borde generalmente necesitan ser más pequeños que los modelos desplegados en Cloud, y consecuentemente requieren equilibrar para compensar aspectos como la complejidad y tamaño de los modelos, la frecuencia de actualización y la baja latencia.

Para convertir un modelo entrenado a un formato que funciona en el “borde”, los modelos se someten a un proceso conocido como “quantization” donde los pesos aprendidos son representados con menor número de bytes. Tensorflow por ejemplo, usa un formato llamado TensorFlow Lite.

Sin embargo, en ocasiones, no es necesario emplear un modelo tan complejo para dar respuesta a las peticiones que se están realizando. En estas situaciones, es posible empezar con un modelo “menor” desplegado en el borde y solo en aquellos momentos en los que sea necesario se ejecutará con un “Trigger” otro modelo más complejo desplegado en Cloud.

Un claro ejemplo de esto, se muestra en la siguiente imagen, con un modelo más simple para la detección de si un sonido pertenece a un instrumento musical, y solo en estos casos, se invocará a un modelo más complejo para identificar qué tipo de instrumento.



## 14. Patrón Reproducibility Design

Este apartado se centra en todas aquellos criterios que deben ser tenidos en cuenta para que un modelo de Machine Learning sea reproducible durante su entrenamiento. Cubre los patrones 21,22 y 24 del libro.

### Transform

En el momento de la inferencia de un modelo, tenemos que saber que características fueron entrenadas y como deberían ser interpretadas y los detalles de las transformaciones que fueron aplicadas. Por ejemplo, si transformamos los días de la semana con una librería u otra puede que el número “1” corresponda al inicio o final de la semana.

La solución para todo esto, es capturar las transformaciones aplicadas para convertir los inputs del modelo usando la cláusula TRANSFORM (En BigQuery) de tal manera que se asegure que las mismas transformaciones son realizadas durante el “predict” del modelo. Para realizar esto en Tensorflow y Keras, disponemos del concepto “feature column”.

Su implementación en Keras se puede realizar tanto con “Preprocessing layers” como con “Lambda layers”. A continuación, se muestra un ejemplo partiendo de 4 variables (features) : Dropoff Latitude, Dropoff Longitude, Pickup Latitude y Pickup Longitude de un problema para predecir el precio del viaje.

Por otro lado, hay que tener en cuenta que si el procesamiento que se quiere realizar debe tener en cuenta el dataset completo para realizar métricas (medias, desviaciones etc), es preferible el uso de tf.transform en vez de Lambda.

Definición de los input:

```
inputs = {
    colname : tf.keras.layers.Input(
        name=colname, shape=(), dtype='float32')
    for colname in ['pickup_longitude', 'pickup_latitude',
                    'dropoff_longitude', 'dropoff_latitude']
}
```

Creación de un diccionario con las variables escaladas aplicando Lambda:

```
transformed = {}
for lon_col in ['pickup_longitude', 'dropoff_longitude']:
    transformed[lon_col] = tf.keras.layers.Lambda(
        lambda x: (x+78)/8.0,
        name='scale_{}'.format(lon_col)
    )(inputs[lon_col])
for lat_col in ['pickup_latitude', 'dropoff_latitude']:
    transformed[lat_col] = tf.keras.layers.Lambda(
        lambda x: (x-37)/8.0,
        name='scale_{}'.format(lat_col)
    )(inputs[lat_col])
```

Creación de nuevas variables aplicando operaciones con Lambda:

```
def euclidean(params):
    lon1, lat1, lon2, lat2 = params
    londiff = lon2 - lon1
    latdiff = lat2 - lat1
    return tf.sqrt(londiff*londiff + latdiff*latdiff)
transformed['euclidean'] = tf.keras.layers.Lambda(euclidean, name='euclidean')([
    inputs['pickup_longitude'],
    inputs['pickup_latitude'],
    inputs['dropoff_longitude'],
    inputs['dropoff_latitude']
])
```

Se realiza la unificación de todas las variables:

```
dnn_inputs = tf.keras.layers.DenseFeatures(feature_columns.values())(transformed)
```

Por último, se construye el modelo:

```
h1 = tf.keras.layers.Dense(32, activation='relu', name='h1')(dnn_inputs)
h2 = tf.keras.layers.Dense(8, activation='relu', name='h2')(h1)
output = tf.keras.layers.Dense(1, name='fare')(h2)
model = tf.keras.models.Model(inputs, output)
model.compile(optimizer='adam', loss='mse', metrics=['mse'])
```

### **Repeatable Splitting**

Existen multitud de tutoriales que sugieren la división de los datos de una forma Random, sin embargo, este enfoque falla en muchas situaciones del mundo real, ya que es raro que los registros del dataset sean completamente independientes.

Por ejemplo, para predecir los retrasos de un vuelo, los retrasos ocurridos en el mismo día estarán altamente correlacionados unos con otros. Este tipo de situaciones, obligan a la realización de técnicas que permitan reproducir los entrenamientos y que además aprovechen el valor de los registros previos.

En primer lugar, debemos identificar una columna que capture la relación de correlación entre filas. En el ejemplo anterior de retrasos aéreos, ésta, sería la columna de “fecha”. A esta columna, podemos aplicar un “hash” con la ayuda de la función “FARM\_FINGERPRINT” de tal manera que encuentre un % de filas (80% por ejemplo para Train) que cubra días completos, y que sea reproducible (mismo resultado siempre) indicándole un día en concreto.

Otras situaciones de interés pueden ser:

- Si las filas no tienen correlación, podemos simplemente aplicar un Random.
- Si estamos trabajando con series temporales (no estacionales), aplicamos separaciones secuenciales, dejando test y Validation para “futuros”.
- Si los registros poseen estacionalidades como el Forecast climático, y los días previos no están tabulados (como otras variables), se podría utilizar como train los 20 primeros días de cada mes, los siguientes 5 como Validation y los últimos 5 como Test.
- Si los Datos están muy desbalanceados (Por ejemplo, antes de las 6 am hay muy pocos retrasos), debemos procurar que todas las franjas queden representadas en cada uno de los conjuntos.

Es muy importante una separación correcta de los datos ya que en ocasiones un rendimiento pobre del modelo puede deberse a que se han perdido las correlaciones con registros previos o no ha sido entrenado con ciertos segmentos de los datos.

### **Windowed Inference**

Hace referencia a la necesidad de proporcionar a los modelos información pasada, o de variables agregadas sin la lectura completa del Data Warehouse.

Una solución para ello es la lectura del archivo histórico definiendo ventanas. Por ejemplo, en el problema anterior de retraso de vuelos, realizar la lectura de las 2 horas previas. A partir de este Dataset, realizar el cálculo de agregados en ventanas menores (10 minutos, por ejemplo), y realizar el “predict” a partir de este Dataset generado.



## 15. Patrón Bridged Schema

En ocasiones se construyen modelos, y con el paso del tiempo se obtiene mayor granularidad en los datos o nuevas variables que podrían aportar mucho valor. Este apartado intenta cubrir una serie de técnicas para reacondicionar los modelos.

Por ejemplo, asumamos que uno de los inputs de los modelos es el tipo de pago. En el histórico esta variable había sido guardada como “cash” o “card”. Sin embargo, con la actualización de los datos ahora se proporciona mayor granularidad y se especifica el tipo de tarjeta: “gift card”, “debit card” o “credit card”. También debemos suponer que no podemos entrenar un modelo únicamente con los nuevos datos ya que la cantidad es pequeña y la calidad de un modelo está fuertemente relacionada con el volumen de datos que se usó para su construcción.

La solución es “puentear” del esquema de datos antiguo al nuevo esquema. Existen dos enfoques principales para realizar esta tarea: Método probabilístico y **Método Estático**. Este último, es el enfoque más **recomendado** por los autores del libro.

### Método Probabilístico

Imagina que estimamos de los datos nuevos de training que de cada transacción de tipo “card”, el 10% pertenecen a “gift”, el 30% a “debit” y el 60% a “credit”. Con estas probabilidades y suponiendo que se mantienen en el histórico, para completar los registros históricos podríamos generar en cada época para cada registro un valor del 1 al 100, y según el valor obtenido, atribuir la categoría correspondiente.

### Método Estático

Las variables categóricas se les aplica normalmente one-hot Encoding y se realiza la media. Si mantenemos las mismas proporciones comentadas en el método anterior, al realizar la media de los registros de “card”, el resultado sería [0, 0.1, 0.3, 0.6] siendo 0 el valor para “cash”.

Para puentear pasado y presente, éste sería el valor proporcionado para todos los registros históricos de tipo “card”, mientras que los nuevos registros tendrán como valor 0 todas las categorías, excepto la que realmente les aplique. Por ejemplo, una transacción de “debit card” en los registros nuevos, podría ser [0, 0, 1, 0], si la tercera posición fuese “debit”.

Este es el enfoque recomendado, ya que además de ser computacionalmente más rápido, el valor proporcionado para el histórico está menos granularizado, pero es “correcto”.

Otra situación a la que podríamos enfrentarnos, es la aparición de nuevas variables que antes no estaban siendo consideradas por el modelo. Si tenemos nuevas ‘features’ que queremos empezar a usar inmediatamente la primera acción es rellenar con missing values estas features y para realizar la imputación se recomiendan los siguientes enfoques:

- Valor medio de la característica si es numérica y está uniformemente distribuida.
- Valor mediano si es numérica pero la distribución no es uniforme.
- Valor mediano de la variable si es categórica y su valor es ordenable (1,2, etc).
- El modo de la característica si es categórica y no ordenable.
- La frecuencia de la característica si es booleana la variable. (Por ejemplo, si es un dataset de lluvia y ocurre el 2% de los días, podríamos sustituir el valor por 0.02)

## 16. Patrón Workflow Pipeline

En este patrón, se maneja el problema de crear un pipeline de inicio a fin, que sea reproducible para su contenerización y orquestación en un proceso de Machine Learning.

En los últimos años, las aplicaciones monolíticas han sido reemplazadas en favor de arquitecturas basadas en microservicios donde las piezas individuales de la lógica de negocio son desplegadas como un paquete aislado de código. Con microservicios, una aplicación grande es dividida en más pequeñas, siendo más gestionables, así que los desarrolladores pueden construir, debuggear, y desplegar piezas de forma independiente.

Cuando un proceso de Machine Learning escala, el Workflow alcanza un tamaño tal, que diferentes personas o grupos en una organización pueden ser responsables de diferentes pasos. Por tanto, necesitamos realizar este flujo con una serie de estrategias para que diferentes personas puedan realizar pruebas y ejecuciones de forma independiente.

Para manejar este problema, podemos hacer cada paso del flujo un servicio separado y contenerizado. Los contenedores, garantizan que seremos capaces de ejecutar el mismo código en diferentes entornos y que veremos un comportamiento consistente. Estos pasos contenerizados, de forma unida forman un “Pipeline”.

**Existen diversas herramientas que permiten la creación de Pipelines con ambos enfoques: On premise y Cloud, como : Cloud AI Platform Pipelines, TensorFlow Extended (TFX), Kubeflow Pipelines (KFP), MLflow y Apache Airflow.** Una de las ventajas de crear un pipeline con TFX o Kubeflow es que podemos extrapolar a Google u otro distribuidor Cloud.

En TFX, los pasos mencionados son conocidos como componentes. y cada uno de ellos tiene una finalidad determinada (Por ejemplo, el primero de ellos se destina a tomar datos de una fuente externa : **ExampleGen**). El siguiente paso de este flujo es la validación de datos (**StatisticsGen**) que permite generar resúmenes estadísticos de los datos proporcionados por ExampleGen. Utilizando el output de **SchemaGen**, **ExampleValidator**, realiza la detección de anomalías, chequea la existencia de data drift o potenciales distribuciones sesgadas. El componente **Transform** realiza con la salida de los componentes anteriores la transformación de las variables y creación de nuevas. Una vez que los datos están listos, mediante el uso del componente **Trainer** y el componente **Pusher** se realiza el entrenamiento y despliegue de los modelos respectivamente. Además de los mencionados, existen otros componentes que pueden ser usados según las necesidades del problema.

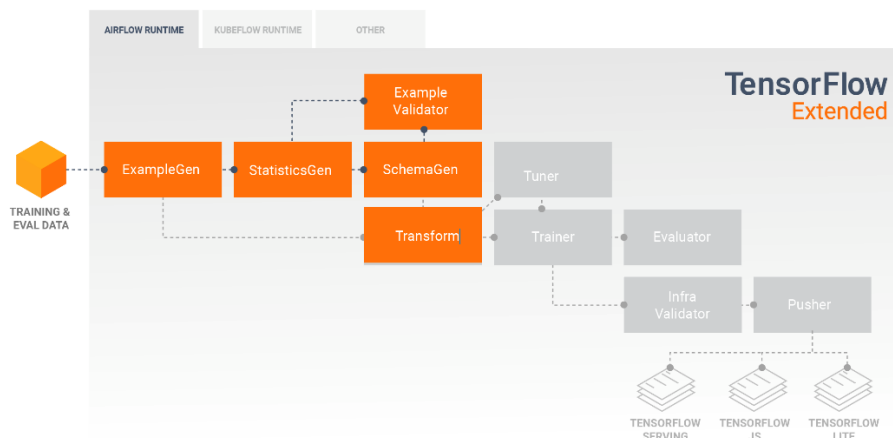
Para la ejecución de estos Pipelines en Cloud, es necesario paquetizar el flujo como un contenedor de Docker, si usamos Google tras esto hospedarlo en Google Container Registry (GRC), y por último crear el Pipeline con TFX CLI:

```
tfx pipeline create \  
  --pipeline-path = Kubeflow_dag_runner.py \  
  --endpoint = 'your-pipelines-dashboard-url' \  
  --build-target-image = 'grc.io/your-pipeline-container-url'
```

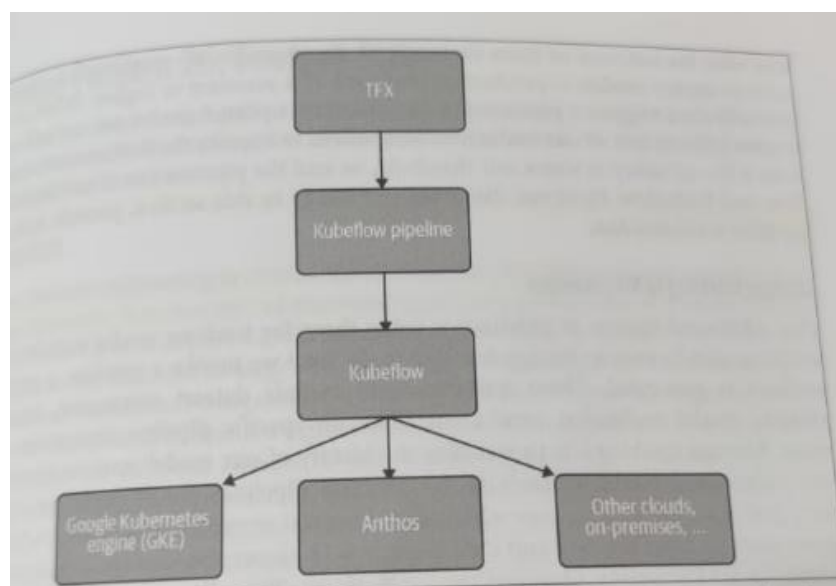
Para invocar el recurso creado:

```
tfx run create --pipeline-name = 'your-pipeline-name' --endpoint = 'pipeline-url'
```

También es posible crear componentes personalizados en TFX (en vez de usar los preconstruidos), o transformar funciones de Python como componentes por sí mismos. Para realizar esto último, solo es necesario con usar el decorador `@component`.



Para la orquestación de los Pipelines tanto TFX como KFP ejecutan sobre Kubeflow, una plataforma para la ejecución de soluciones de ML containerizadas en Kubernetes.



## 17. Patrón Model Versioning

Desplegar actualizaciones a los modelos de producción afectará inevitablemente al comportamiento sobre los nuevos datos, y esto supone un reto que hay que controlar. Cuando realizamos estas actualizaciones por la existencia de un dataset con nuevos datos, aunque la precisión del modelo aumenta (con respecto a la precisión que marcaba la necesidad de reentrenamiento), la precisión alcanzada en datos antiguos disminuye ligeramente.

Para manejar esto, necesitamos una solución que permita a los usuarios seleccionar versiones anteriores de nuestro modelo si ellos lo prefieren.

Para manejar de una forma sencilla las actualizaciones de un modelo, se pueden desplegar varias versiones con REST Endpoints diferentes. Los usuarios que confíen más en las versiones antiguas serán capaces de usar este servicio. Si podemos dividir los usuarios en distintos grupos (basados en el uso de la app que tengamos) podemos ofrecer a cada grupo la versión ajustada a sus preferencias.

A continuación, mostramos un ejemplo de despliegue en Cloud AI Platform suponiendo la existencia de un modelo cuyo archivo exportado es “model.bst”, ubicado en un Cloud Storage Bucket:

```
gcloud ai-platform versions create 'v1' \
  --model 'flight_delay_prediction' \
  --origin gs://your-gcs-bucket \
  --runtime-version=1.15 \
  --framework 'XGBOOST' \
  --python-version=3.7
```

Con este modelo desplegado, ahora es accesible mediante el Endpoint: /models/flight\_delay\_prediction/versions/v1 con una petición HTTPS, y para realizar predicciones podemos enviar los input en el formato que el modelo espera (dentro de un JSON, por ejemplo).

```
gcloud ai-platform predict --model 'flight_delay_prediction' \
  --version 'v1' \
  --json-request 'input.json'
```

Azure y AWS tienen servicios similares de versionado disponibles. En Azure el despliegue de modelos está disponible con Azure Machine Learning y en AWS estos servicios se realizan con SageMaker. También es posible usar herramientas Open Source como TensorFlow Serving cuyo uso está recomendado empleando Docker o MLflow.

## 18. Patrón Explainable Predictions and Fairness Lens

Es importante que las predicciones del modelo puedan ser explicadas y que además sus decisiones sean “justas” sin crear ningún tipo de discriminación frente a algún grupo social. Necesitamos una forma de entender como los modelos están actuando y que decisiones están tomando para generar una predicción.

Aunque es tentador asignar significado según el valor de los pesos aprendidos en regresiones lineales o haciendo modelos de árboles, debemos ser extremadamente cautelosos cuando lo hagamos ya que las conclusiones que dibujemos pueden ser incorrectas.

El tipo de métodos que vamos a comentar son conocidos como “feature attributions”. Estos métodos se dirigen a atribuir al output del modelo un valor para cada característica, indicando, qué importancia ha tenido en la predicción. Hay dos tipos de “feature attributions”: Instance-level que se ocupa de realizar este proceso instancia a instancia (para cada predicción individual) y Global que agrega estas conclusiones para indicar cómo se comporta el modelo completo.

A continuación, explicamos los tres principales métodos existentes en la actualidad:

## Sampled Shapley

Basado en el concepto de Shapley Value, este enfoque determina la contribución de una variable o característica, analizando como añadiendo y eliminando dicha característica se ve afectada la predicción. Para su uso con Python podemos emplear **la librería SHAP** que permite su uso en multitud de tipos de modelos.

```
e = shap.DeepExplainer(model, images)
shap_values = e.shap_values(test_images)
```

Las variables “images” y “test\_images”, podrían ser sustituidos por “x\_train” y “x\_test” en datos tabulares.

## Integrated Gradients (IG)

Usando un modelo base predefinido, IG calcula las derivadas (gradientes) desde el modelo base al input específico. Este enfoque es recomendado para imágenes no naturales como las tomadas en un laboratorio o empresa farmacéutica, ya que se enfoque en las diferencias pixel a pixel (Si queremos detectar un tumor necesitamos esta granularidad). Otro enfoque basado en IG como XRAI normalmente trabaja mejor en las imágenes naturales ya que da más importancia a un mayor nivel de granularidad (Para detectar si un animal es un perro o un gato, la decisión no será por un pixel).

## Model Baseline

Este enfoque genera un valor “mediano” evaluando el modelo, con el valor de la mediana para cada una de las variables que actúan como Input. A partir de aquí, en cada una de las predicciones se calcula para cada parte de la ecuación (Con cada parte nos referimos a variable\*coeficiente), que unidades de cambio supone respecto a ese valor “mediano” que se generó al inicio. (Por ejemplo, si nuestro valor mediano era 15, y el modelo predice 21, justificar que variables han causado que el valor sea superior en 6 unidades).

Debemos tener en cuenta que el “Bias” en los datos y humanos es inevitable y no siempre tiene que ser malo. El Bias llega a ser dañino cuando afecta a diferentes grupos de personas de forma diferente, esto es conocido como “Bias problemático”. Esto puede ocurrir cuando no se incluye una distribución equitativa de edades, razas, géneros religiones o orientaciones sexuales entre otras características.

Un enfoque común que puede llevar a problemas es la eliminación de estas variables que causarán el problema, sin embargo, este bias, puede verse reflejado en otras variables que están fuertemente correlacionadas con la eliminada, esto es conocido como “implicit o proxy bias”.

Por tanto, ya que los modelos son la imagen de los datos que han sido proporcionados durante la fase de entrenamiento, es importante que se consiga tanto eliminar el “data distribution bias” en el que el problema se debe a que no disponemos de suficientes muestras para ciertos grupos sociales que usarán el modelo, como el “data representation bias”, en el que no existe una representación equitativa de casos positivos y negativos para ciertos grupos sociales.