UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

# Simulación de datos de sensores industriales

**TRABAJO DE FIN DE MÁSTER**

Máster en Big Data Analytics

*Autor:* Pedro Henrique Mano Figueiredo Fernandes
*Tutor:* Francisco Sánchez Cid

Curso 2015-2016

# Abstract

Los entornos industriales hacen uso de sensores para obtener mediciones de varios factores en su cadena de producción. En las fases iniciales, la cantidad de datos generados es muy reducida, por lo que no es significativa para permitir la aplicación de técnicas de Big Data. Estas técnicas pueden descubrir muchos patrones en los datos, útiles para detección de anomalías o predicción futura. El proyecto descrito en este documento busca simular nuevos datos a través de un pequeña cantidad de datos generados por sensores industriales. La aplicación de estas técnicas se extiende a todo el contexto de proyectos IoT (*Internet of Things*), donde los sensores representan la principal fuente de datos.

**Palabras clave:** Machine learning, Big Data, PCA

# Abstract

The industrial environments make use of sensors for acquiring metrics on several variables in their production pipeline. In the initial phases, the generated data has very low volume, making it hard to apply Big Data techniques. These techniques can bring much insight over the data, useful for anomaly detection or future prediction. The project described in this document seeks the simulation of new data using a small amount of data generated by industrial sensors. The application of such techniques can be extended to the entire context of IoT (*Internet of Things*), where sensors represent the main source of data.

**Key words:** Machine learning, Big Data, PCA

# Índice general

# Índice de figuras

# Índice de tablas

# Índice general

# CAPÍTULO 1
# Introducción

Este proyecto tiene como fuente de datos uno o varios sensores de un mecanizado industrial de inyección de plástico. Los sensores efectúan mediciones de varios factores con una regularidad temporal, generando así muestras de datos en determinados intervalos de tiempo. Cuando se trata de sensores, es normal que los datos no tengan la calidad necesaria para aplicar técnicas matemáticas. Hay que tener en cuenta que la frecuencia de las mediciones no es necesariamente constante, que pueden existir interrupciones, ruido y redundancias. Además, en el ámbito de IoT (*Internet of Things*), se añade la inherente conectividad de los aparatos como factor de complejidad.

Las técnicas de Big Data, en particular *Machine Learning*, necesitan gran cantidad de datos para poder inducir modelos matemáticos que expliquen esos datos. Cuanto más datos mejor, para un aprendizaje más robusto y fiable. Sin embargo, al principio de los proyectos, la cantidad de datos recolectados suele ser pequeña y insuficiente para poder extraer conocimiento significativo de los mismos. El objetivo de este proyecto es aprender y simular el comportamiento de uno o varios sensores a través de una cantidad reducida de datos recolectados en los mismos.

Una vez preparado el *dataset*, es necesario aprender un modelo matemático que lo describa. Las técnicas usadas para ese efecto son del ámbito de *Machine Learning*.

## 1.1 Estructura del documento

Este documento se estructura de la siguiente forma: la descripción del problema; el estado del arte; un análisis sobre el *dataset* y respectivas transformaciones para adecuar los datos; las soluciones propuestas y respectiva base teórica; la experimentación, resultados y discusión; conclusiones del trabajo realizado y posibles mejoras; y termina con las fuentes bibliográficas que han servido de base del estudio.

# CAPÍTULO 2
# Descripción del problema

El paradigma de IoT (*Internet of Things*) tiene mucha presencia en la industria. Se utilizan sensores para controlar determinados factores en las cadenas de producción. Los datos generados son de gran importancia. Con el debido tratamiento y análisis, estos datos pueden ayudar descubrir conocimiento nuevo. En el entorno industrial, ese conocimiento se puede traducir en la anticipación de anomalías o mejoras de rendimiento de las máquinas, con todo el beneficio que eso conlleva.

La extracción de conocimiento de los datos se hace con técnicas estadísticas de *Machine Learning*. Estas técnicas están relacionadas con Big Data por la simple regla de que cuantos más datos mejor. Los modelos matemáticos que explican los datos son más robustos si hay mucho volumen de datos. Sin embargo, en las etapas iniciales de implantación de IoT no hay mucha cantidad de datos, lo que dificulta la aplicación de *Machine Learning*.

Hay que tener en cuenta que los datos de sensores suelen tener problemas de calidad, sobre todo relacionados con ruido, redundancias o frecuencia de muestro. Por ejemplo, si un sensor mide la temperatura de una máquina, esa medición sufre interferencias del ambiente, por lo que la medición no es completamente objetiva. Ese efecto se conoce por ruido. Cuanto a las redundancias, pueden suceder en casos de pérdida de conectividad, en que el sensor vuelve a dar una métrica aunque la haya dado ya anteriormente. Esto sería, también, un caso de cambio de la frecuencia de muestreo, que se traduce en irregularidades de la serie temporal asociada a los datos.

El objetivo de este proyecto es aprender y simular datos de sensores industriales. Los nuevos datos, de mucho mayor volumen, podrán ser usados para probar y depurar los algoritmos de detección de averías y de predicción desarrollados para entornos Big Data.

# CAPÍTULO 3
# Estado del Arte

Los problemas de ruido y redundancias en los datos son comunes y están bastante estudiados. Para solucionar ese problema, las técnicas de reducción de dimensionalidad, en que los datos son reducidos a lo esencial, suelen tener buenos resultados. PCA (*Principal Component Analysis* es la técnica más usual. PCA se usa para descomponer un conjunto de datos multivariante en un grupo de componentes ortogonales que mejor explican la varianza. El ajuste a los datos es de tipo lineal.

Las primeras componentes son las que mayor varianza explican y por eso se llaman principales. El orden de las componentes es relevante, la primera es la más explicativa, la segunda la que más explica quitando la primera, y así sucesivamente. Así que es de esperar que lo que están explicando las últimas sea, realmente, el ruido de la señal. El artículo 'A Tutorial on Principal Component Analysis' deja patente esa evidencia en el objetivo de filtrar el ruido y redundancias para obtener la dinámica importante de los datos.

Aunque PCA no tenga capacidades de predicción (pertenece a la categoría de *unsupervised learning*), se puede usar para simular nuevos datos, como se defiende en el artículo 'The Truth About Principal Component and Factor Analysis'. Aparte de las componentes que explican los datos, PCA obtiene también las proyecciones a lo largo de las componentes. Se pueden aprender las series de los *scores* de las proyecciones, reemplazar por nuevos valores y luego transformar de vuelta a un vector de características originales.

Las observaciones de los sensores tienen una implícita ordenación cronológica (y así lo tendrán también las componentes principales). El estudio de esa relación se denomina análisis de series temporales. Con este análisis ahí se puede llegar a explicar mejor la serie y hacer predicciones sobre el futuro de la misma.

Los modelos *Exponential smoothing* y *AutoRegressive Integrated Moving Average* (ARIMA) son los más usados para predicción de series temporales. *Exponential smoothing* se basan en la descripción de tendencia y de variación estacional. Pero, en general, observaciones sucesivas suelen tener dependencia entre si, como se describe en el libro 'Introduction to Time Series Analysis and Forecasting'. Para incorporar esta dependencia se usan los modelos ARIMA. Los modelos ARIMA son un flexible y poderoso método para análisis de series temporales y predicción. A lo largo de los años, se vienen usando con éxito en varios problemas de investigación y en la práctica.

# CAPÍTULO 4
# Solución propuesta

La solución propuesta para la simulación de datos empieza con la proyección de los datos del espacio multivariante original a un espacio ortogonal más reducido. Esta técnica de *Machine Learning* se conoce por PCA (*Principal Component Analysis*). El nuevo espacio (con menos dimensiones) contiene las componentes que mejor explican los datos, por lo que se eliminan ruido y redundancias.

Para cada componente, se efectúa la generación de nuevos valores con distribución Gaussiana. Esta simulación puede ser de miles o millones de nuevos puntos - no es una tarea computacional exigente y además tendrá utilidad más adelante. La simulación Gaussiana servirá como base del estudio, una vez que, al ser invertida, genera una población que es idéntica a la original. La comprobación de dicha similitud de poblaciones se hace a través del estadístico T-cuadrado de Hotelling. Y esta será nuestra hipótesis nula.

Los datos originales tienen un sentido temporal. Las componentes lo tendrán también, así que se procede al análisis de la serie temporal de esas componentes. El modelo propuesto es ARIMA (*Autoregressive Integrated Moving Average*). Los modelos de series temporales se ajustan a los datos para mejor comprensión de estos o para predicción de nuevos puntos en la serie. El objetivo es dotar la simulación de un patrón temporal aprendido previamente.

Teniendo en cuenta que el estudio se sustenta en la hipótesis nula de la simulación Gaussiana, para cada predicción de las varias componentes con marca temporal, se buscará el punto multi-dimensional más cercano en la simulación Gaussiana. El punto elegido se usará para re-alimentar el modelo ARIMA aprendido. Llegados aquí, tenemos nuevos valores simulados de las componentes y además con carácter temporal. Al reproyectar este espacio al espacio original obtenemos nuevos valores (simulados) para las características originales.

La simulación Gaussiana debe tener gran volumen para que la simulación final sea efectiva. Cuantos más puntos tenga mejor, para que la búsqueda encuentre puntos más cercanos. Con un volumen de datos muy grande, los paradigmas de programación convencionales no tienen capacidad de respuesta. Se adoptan herramientas Big Data (*Spark* y *MongoDB*) para permitir escalar la parte de simulación Gaussiana, su almacenamiento y la búsqueda de puntos cercanos.

## 4.1 Reducción de dimensionalidad - PCA

PCA (*Principal Components Analysis*) pertenece a la familia de aprendizaje no supervisado (*unsupervised learning*), de *Machine Learning*, donde no existen etiquetas o clases

a predecir. Los datos de entrenamiento son simplemente observaciones, sin estar clasificados. Las técnicas de *unsupervised learning* se usan para descubrir grupos de similitud (clusters) en los datos; o para determinar la distribución de los datos en el espacio original; o también, que interesa a este estudio, para proyectar los datos en espacios de menos dimensiones.

Pocas dimensiones suelen ofrecer mejor *insight* sobre los datos. Como primera ventaja, permiten la aplicación de técnicas de visualización - resulta muy difícil visualizar datos en más de 3 dimensiones (incluso en 3 dimensiones puede ser difícil). Además, las componentes que no son principales (menos explicativas) suelen ser ruido o redundancia - con esta técnica se pueden atenuar los efectos de estos. La simulación de datos puede usar la ventaja de la eliminación de ruido y redundancia, generando así datos con mayor entidad.

PCA descompone un *dataset* multivariante en un determinado número de componentes ortogonales que son los que más explican la varianza de ese *dataset*. El número de componentes es menor o igual al número de características originales. Por ejemplo, si dos variables son directamente proporcionales, basta con una sola componente para explicar el comportamiento de las dos - esa correlación quedará representada en la matriz calculada por PCA. Si se convierte esa componente al espacio original de dos características, se obtienen los valores originales.

### 4.1.1. Detalle

TODO: explicar con gráficos

En términos matemáticos, PCA es una transformación lineal ortogonal. Consideremos la matriz $X$, constituida por $n$ líneas de observaciones y $m$ características. El objetivo es proyectar los datos a un espacio con dimensionalidad $d < m$, de forma que se maximice la varianza de los datos proyectados. La transformación de PCA se define por la ecuación:

$$Y = X \cdot w \tag{4.1}$$

Que se interpreta de la siguiente forma: una matriz $m \times k$ de pesos (*loadings*) $w$ transforma la matriz $n \times m$ $X$ en la matriz $n \times k$ de componentes principales (*scores*) $Y$.

Para obtener la primera componente, el primer vector de *loadings* $w$ debe maximizar la varianza. Siguiendo el razonamiento del libro "TODO", consideremos el vector $w_1$, que, por conveniencia (y sin pérdida de generalización) debe ser un vector unitario de modo que $w_1^T w_1 = 1$. Cada punto de X, $x_i$, es proyectado para el escalar $w_1^T x_i$. La media de los datos proyectados es $w_1^T \overline{x}$, donde $\overline{x}$ es la media dada por:

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{4.2}$$

Y la varianza de los datos proyectados:

$$\frac{1}{n} \sum_{i=1}^{n} x_i \{ w_1^T \cdot x_i - w_1^T \cdot \overline{x} \} = w_1^T \cdot S \cdot w_1 \tag{4.3}$$

Donde $S$ es la matriz de varianzas-covarianzas definida por:

$$S = \frac{1}{n} \sum_{i=1}^{n} (x_i - \overline{x}) \cdot (x_i - \overline{x})^T \tag{4.4}$$

Ahora maximizamos la varianza de los datos proyectados $w_1^T \cdot S \cdot w_1$ con respecto a $w_1$. Hay que restringir la maximización para prevenir que tienda para infinito. La restricción viene de la normalización $w_1^T w_1 = 1$. Para forzar la restricción introducimos un multiplicador de Lagrange $\lambda_1$:

$$w_1^T \cdot S \cdot w_1 + \lambda_1(1 - w_1^T \cdot w_1) \tag{4.5}$$

Que tiene un punto estacionario cuando:

$$S \cdot w_1 = \lambda_1 \cdot w_1 \tag{4.6}$$

Esto define que $w_1$ es un vector propio de $S$. Si multiplicamos las parte izquierda por $w_1^T$, teniendo en cuenta que $w_1^T w_1 = 1$, vemos que la varianza es dada por:

$$w_1^T \cdot S \cdot w_1 = \lambda_1 \tag{4.7}$$

Así que la varianza será máxima cuando se defina $w_1$ igual al vector propio con el máximo valor propio $\lambda_1$.

### 4.1.2. Singular Value Decomposition - SVD

PCA está muy relacionado con una técnica matemática llamada *Singular Value Decomposition* (SVD), tanto que muchas veces los nombres se usan intercambiados. De hecho, el algoritmo de PCA de *scikit-learn* usa la descomposición SVD de *numpy*. SVD es un método más general de entender el cambio de base.

La representación de la descomposición en valores singulares es:

$$X = U\Sigma V^T \tag{4.8}$$

Donde:

- $U$ es una matriz $n \times n$, en que las columnas son vectores unitarios ortogonales de tamaño $n$.
- $\Sigma$ es una matriz diagonal $n \times m$ de números positivos $\sigma_i$, llamados valores singulares de $X$.
- $W$ es una matriz $m \times m$, cuyas columnas son vectores unitarios ortogonales de tamaño $p$.

La ecuación 4.2 indica que una matriz $X$ puede ser convertida en una matriz ortogonal, una matriz diagonal y otra matriz ortogonal. O, dicho de otra forma, corresponde a una rotación, un estiramiento y otra rotación.

## 4.2 Series temporales - ARIMA

TODO

# CAPÍTULO 5
# Experimentación y resultados

TODO

## 5.1 Dataset

El dataset proporcionado es de reducida dimensión, tiene tan solo 5MB, por lo que la tarea de ingestión no es intensiva. Sin embargo, hay que tratar los datos en bruto antes de empezar a usarlos. El fichero de datos es de texto, así que hay que hacer determinadas conversiones para poder usar tipos más específicos, como sean fechas y números. El primer problema tiene que ver con los formatos de fecha, que no son correctos para el *locale* España en *Windows*, lo que exige una adaptación de los algoritmos de *parsing*, como se describe en el apartado 'Transformación'.

### 5.1.1. Estructura

Un breve análisis del *dataset* en un editor de texto muestra que tiene un formato de campos separados por espacios y tabulaciones. La cantidad de espacios es variable:

```
␣␣␣␣␣␣␣␣Tiempoinicio␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣APHu␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣APVs␣
    ...

␣␣␣␣␣␣␣␣06-oct-2015␣21:57:03␣␣44.6␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣69.3␣...
␣␣␣␣␣␣␣␣06-oct-2015␣21:57:12␣␣45.1␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣69.0␣...
␣␣␣␣␣␣␣␣06-oct-2015␣21:57:21␣␣44.8␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣69.8␣...
␣␣␣␣␣␣␣␣...
␣␣␣␣␣␣␣␣
```

**Listing 5.1:** Ejemplo del *dataset*.

La primera línea contiene un *header* (cabecera), con 15 nombres:

```
Tiempoinicio APHu APVs ACPv ZSx ZUs H7x H1x H2x H6x H3x H4x H5x ACPx Svo
```

**Listing 5.2:** *Header* del *dataset*.

Se puede verificar también que hay una línea vacía después del *header*. El primer campo tiene un formato de fecha/hora y los demás campos tienen formato decimal.

### 5.1.2. Transformación

Para la ingestión y transformación de los datos se han usado librerías muy útiles y con muchas funcionalidades que facilitan bastante esas tareas: en los scripts Python se ha usado el paquete *Pandas* y en R la función *read.csv2* del paquete *utils*.

Las 14 variables decimales no ofrecen problemas en la ingestión del *dataset*. Para asegurar el formato decimal, es conveniente definir el separador decimal como punto ('.'). Con eso es suficiente para un em parsing correcto.

Las fechas son más complejas de procesar. El formato de mes da indicios de estar escrito en castellano: `oct, dic, mar, abr, may, jun`. Sin embargo, en *Windows*, el *locale* España usa un punto ('.') como *standard* en la abreviación de mes, por ejemplo 'oct.'. Así que el *parsing* de fechas tiene que ser ajustado si el sistema operativo es *Windows*. La estrategia ha sido el uso de una expresión regular para añadir el punto ('.') necesario en las abreviaciones de meses, como se puede ver en los siguientes *snippets* Python y R:

```
def parse_date(date_string):
    locale_date_string = re.sub("(.+-)(.+)(-.+)", "\\1\\2.\\3",
        date_string)
    return datetime.strptime(locale_date_string, "%d-%b-%Y␣%H:%M:%S")
```

**Listing 5.3:** *Parsing* de fechas en Python.

```
data$Tiempoinicio <- sub("(\\d+-)(\\w+)(-\\d+\\s\\d+:\\d+:\\d+)",
    "\\1\\2.\\3", data$Tiempoinicio)
data$Tiempoinicio <- as.POSIXct(data$Tiempoinicio, format="%d-%b-%Y␣
    %H:%M:%S")
```

**Listing 5.4:** *Parsing* de fechas en R.

En este caso particular de las fechas, el problema no está en el *dataset realmente*. Los *standards* de fechas varian en cada sistema operativo. La idea de exponer este caso no es más que transmitir la necesidad de ajustar y normalizar los datos cuando provienen de sensores.

El resultado final es un *dataset* estructurado con una marca temporal como índice y 14 características.

### 5.1.3. Distribución de los datos

Las mediciones de los sensores se distribuyen por varios días, no siempre consecutivos. Los días con mediciones son algunos

- 6-9, 12 de octubre de 2015
- 14 de diciembre de 2015
- 7, 8, 11, 14 de marzo de 2016
- 6, 29 de abril de 2016
- 2, 26 de mayo de 2016
- 8, 17, 20 de junio de 2016

En cada día las muestras son dispares y en momentos distintos del día, probablemente por corresponder a distintas pruebas. Así, también las frecuencias de muestreo son

variables, en algunos días parece indicar una frecuencia de 10 segundos pero hay otros donde son de 1 minuto.

## 5.2 PCA

TODO

### 5.2.1. PCA iterativo - NIPALS

Sin embargo, el cálculo de SVD es muy intensivo para matrices grandes, porque calcula la matriz de varianzas-covarianzas para todos los componentes. Esto implica un gran consumo de memoria y CPU. Como alternativa, se propone usar una técnica iterativa llamada NIPALS (*Nonlinear Iterative Partial Least Squares*), que usa el numero de componentes reducido. TODO: explain NIPALS algorithm TODO

### 5.2.2. Test de hipótesis - Estadístico T-cuadrado de Hotelling

TODO

### 5.2.3. Simulación con distribución Gaussiana - hipótesis nula

TODO

## 5.3 ARIMA

TODO

### 5.3.1. Simulación con todo el dataset

TODO

### 5.3.2. Simulación con datos de 1 día

TODO

### 5.3.3. Simulación con búsqueda de puntos cercanos de la Gaussiana

TODO

# CAPÍTULO 6
# Big Data

TODO

## 6.1 MongoDB

TODO

## 6.2 Spark

TODO

# CAPÍTULO 7
# Conclusiones

TODO

CAPÍTULO 8

# Trabajos futuros

# Bibliografía


[1] Jon Shlens. A TUTORIAL ON PRINCIPAL COMPONENT ANALYSIS. Derivation, Discussion and Singular Value Decomposition. Version 1, 25 March, 2003.

[2] Unknown Authors. The Truth about Principal Components and Factor Analysis. 28 September, 2009.

[3] Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.

[4] Principal component analysis (PCA). Consultar http://scikit-learn.org/stable/modules/decomposition.html#principal-component-analysis-pca.