Aprendizaje Automático

Universidad de Granada

Proyecto Final

Occupancy detection

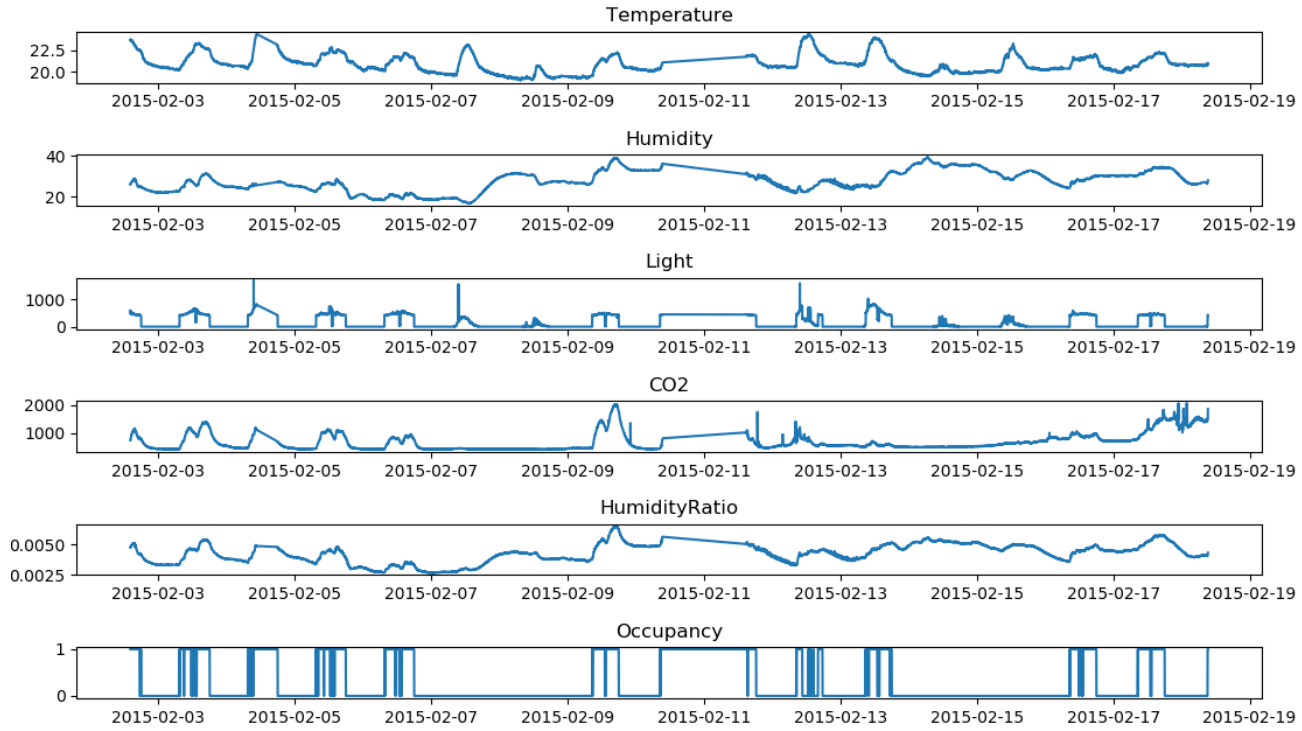
Ignacio Vellido Expósito

Grupo Viernes

Comprensión del problema

Contamos con un problema de clasificación donde nuestro objetivo es determinar si una habitación está ocupada. Para ello se nos proporcionan tres conjuntos de datos (uno de entrenamiento y dos de test), en el que existen las siguientes columnas:

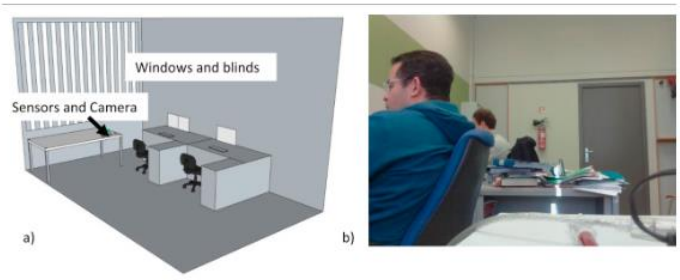
- **Identificador de la muestra**
- **Fecha de la medida**: Siguiendo el formato "año-mes-día hora:minuto:segundo".
- **Temperatura**: En grados centígrados.
- **Humedad relativa**: En tantos por ciento.
- **Cantidad de luz**: Medida en luxs.
- **Dióxido de carbono (CO2)**: En partes por millón.
- **Ratio de humedad**: En kilogramos de agua por kilogramos de aire.
- **Presencia**: 1 en el caso de que la habitación se encontrase ocupada, 0 si no.



Variación de la ocupación, ver bibliografía, punto 4

Leyendo el artículo sobre los datos encontramos más información sobre cómo se realizaron las muestras *(ver bibliografía, punto 2):*

"*The data was recorded during winter in Mons, Belgium during the month of February. The room was heated by hot water radiators that kept the room above 19 °C. In order to estimate the difference in occupancy detection accuracy given by the models, they are tested for data sets when the office door is open and closed. The readings were recorded at time intervals of 14 s or 3 to 4 times per minute, and then averaged for the corresponding minute.*"



Download high-res image (335KB) Download full-size image

Fig. 2. (a) Room sketch showing the position of the sensors and the position of the occupants (b) Example of one of the pictures from the digital camera used to establish ground occupancy.

Lo que nos indica que las muestras se realizaron cada minuto en varios días diferentes, todos durante el mismo mes. También vemos que la temperatura no baja de los 19ºC.

De esta información podemos sacar unas ideas preliminares:

- Por la disposición de la imagen, podemos intuir que en horas con poca luminosidad (tarde o noche) la cantidad de luz medida puede indicarnos si la habitación está ocupada, pues se debería a que alguien ha encendido una luz artificial.
- Por otro lado, un bajo nivel de luminosidad nos debería dar mucha probabilidad de no presencia.
- Las mediciones se realizan únicamente en febrero, por lo que la parte del año y del mes en la fecha son irrelevantes para el aprendizaje.

Codificación y preprocesado de datos

Primeramente, como el identificador no es un dato relevante para el problema, se elimina.

Variable *fecha*

Para poder trabajar con la fecha, al provenir inicialmente de una cadena de texto, la separamos en sus diferentes componentes numéricas (año, mes, día, hora, minuto y segundo). Puesto que el año y el mes no varían, se eliminan.

El resto de variables sobre la fecha son cíclicas, por lo que debería ser apropiado aplicarle una función seno/coseno para representar, por ejemplo, que las once de la noche y las dos de la mañana están próximas entre sí.

Es posible que, en nuestro problema, al haberse realizado las muestras en un corto período de tiempo (cada minuto durante pocos días), estas variables no sean significativas para poder predecir en casos más generales, pero sigo considerándolo una buena práctica a seguir cuando nos encontremos con este tipo de datos.

```
# Aplicamos función seno y coseno
dias_seno = np.sin(dias*(2.*np.pi/30))
dias_coseno = np.cos(dias*(2.*np.pi/30))

horas_seno = np.sin(horas*(2.*np.pi/24))
horas_coseno = np.cos(horas*(2.*np.pi/24))

minutos_seno = np.sin(minutos*(2.*np.pi/60))
minutos_coseno = np.cos(minutos*(2.*np.pi/60))

segundos_seno = np.sin(segundos*(2.*np.pi/60))
segundos_coseno = np.cos(segundos*(2.*np.pi/60))
```

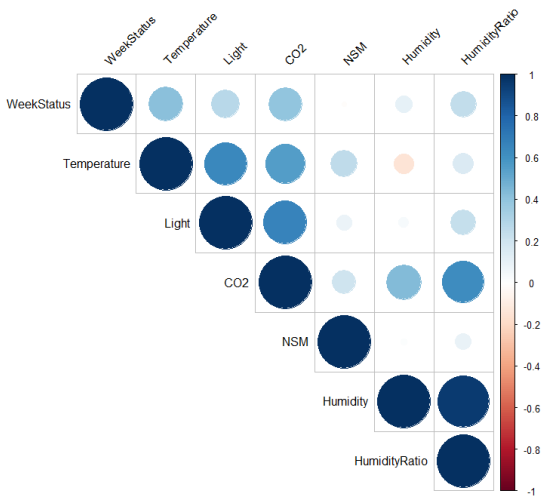
Estandarización

Las distintas características de los datos varían en diferentes rangos, por lo que en principio aplicamos estandarización, restando la media y dividiendo por la varianza.

```
# Aplicamos normalización
scaler = preprocessing.StandardScaler().fit(data_x)
data_x = scaler.transform(data_x)
```

PCA

En principio no se ve interés en reducir la dimensionalidad de los datos, ya que, aunque existe cierta correlación, contamos con un número bastante pequeño de variables.



Matriz de correlación, ver bibliografía, punto 3

Train/test

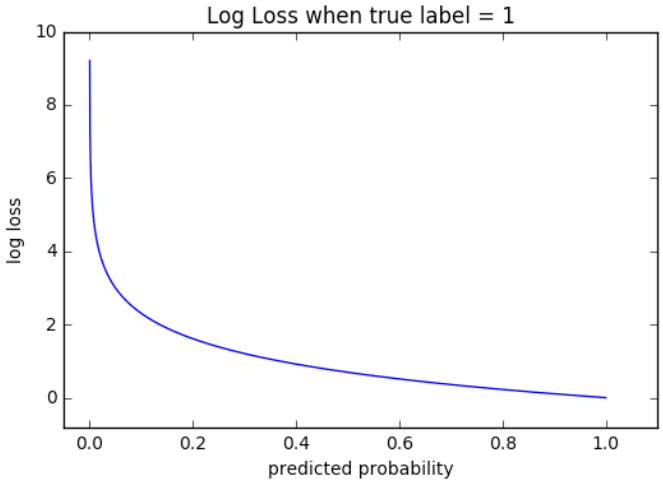
Por último, la división en entrenamiento y test que nos proporcionan no me parece adecuada, ya que esta no es arbitraria, sino que coge unos rangos intermedios de las muestras. Por tanto, prosigo con la idea de tener un único conjunto de datos y un único de validación elegidos de manera aleatoria, en principio con unas particiones del 80 y 20 por ciento.

```
# Separamos en training y test
train_x, test_x, train_y, test_y = train_test_split(data_x, data_y, test_size=0.2)
```

Este conjunto de validación permanece intocable en el resto de apartados (no se utiliza en cross-validation), usándose únicamente para evaluar los modelos.

Selección de la función de pérdida

Para los modelos probados se utiliza cross-entropy, que, siguiendo la implementación de las funciones de *Sklearn*, se utiliza negada. Se elige por ser una función continua que facilita la convergencia de los algoritmos.



Para SVM, por su funcionamiento en sí, se hace la búsqueda del máximo margen en los datos.
De la misma manera, AdaBoost minimiza el error exponencial.

Discusión sobre regularización

Hemos visto que aplicar métodos de regularización es prácticamente obligatorio, pues en caso de que no aporte nada de forma positiva se ignorará la variable de regularización, y en casos favorables podemos evitar el sobreajuste en la muestra. Dependiendo del modelo se regularizará de una manera u otra.

Selección de técnicas

Se realiza el ajuste con diferentes técnicas: Regresión Logística, Support Vector Machine, Random Forest y Boosting.

En todos los modelos se estiman los hiperparámetros mediante *cross-validation*, concretamente con un número de particiones igual a 10. Se elige este número, aparte de por ser el recomendado en la literatura, por coger conjuntos suficientemente grandes que hacen que no sea muy dependiente de la muestra. Además, contamos con un conjunto de datos bastante amplio que nos permiten elegir particiones de este tamaño.

Métricas usadas

Se utiliza el porcentaje de aciertos en la muestra (precisión) y el área bajo la curva.
También se calcula la sensibilidad y especificidad (corresponden a los dos primeros valores de la columna *recall* en la matriz de resultados), y se muestra la matriz de confusión para la predicción de test, de manera que podamos tener un número exacto de los falsos positivos/negativos de nuestro modelo.

Ajuste con Random Forest

El número de características que se seleccionan se mantiene al visto en clase, $m = \sqrt{p}$

Mediante CV se ajustan dos valores:

- El número de árboles.
- La profundidad máxima de los árboles, de manera que podamos aplicar distintos grados de regularización.

```
parameters = {
              'n_estimators': [50, 100, 200],
              'max_depth': [10, 100, 200, 1000]
             }
```

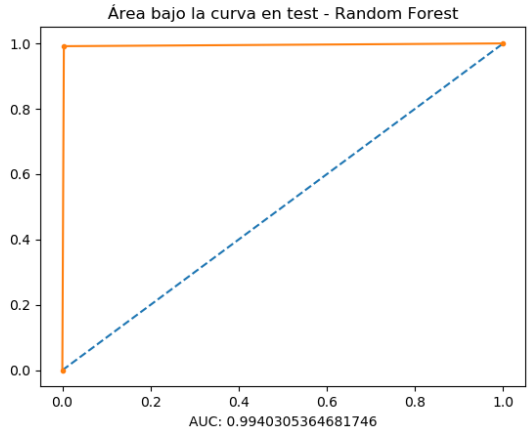
Para el código de entrega, ya que con la búsqueda de rejilla el tiempo de cómputo es excesivamente alto, se reduce el rango a los valores con mejores resultados.

```
parameters = {
              'n_estimators': [100, 200],
              'max_depth': [100, 200]
             }
```

Con estos parámetros obtenemos unos resultados perfectos en la muestra, siendo un indicativo de posible sobreajuste. Pero si comparamos con los resultados en test vemos que estos también son bastante positivos.

```
Ajuste con Random Forest

Mejor parámetro:  {'max_depth': 100, 'n_estimators': 200}
Random Forest train error:  -0.004033802624025747
Random Forest test error:  -0.012067912036035007

Random Forest train accuracy:  1.0
Random Forest test accuracy:  0.9956225680933852

Random Forest train area bajo la curva:  1.0
Random Forest test area bajo curva:  0.9949634604970963

              precision    recall  f1-score   support

  No ocupada       1.00      1.00      1.00      3155
     Ocupada       0.99      0.99      0.99       957

   micro avg       1.00      1.00      1.00      4112
   macro avg       0.99      0.99      0.99      4112
weighted avg       1.00      1.00      1.00      4112


Matriz de confusión:
 [[3143   12]
 [   6  951]]
```

Esta situación también se da con el resto de modelos, y más adelante se hace un análisis de las posibles causas que pueden originar estos resultados.



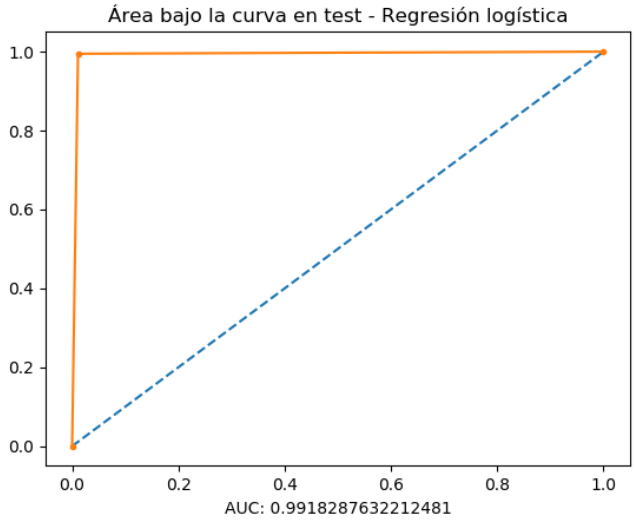
Ajuste con Regresión Logística

En este caso el hiperparámetro estimado con *CV* es C, la inversa del parámetro de regularización.
Se prueba con un rango grande, calculando tanto con alto grado de regularización como bajo.

```
parameters = {
                'C': [1000, 100, 10, 1, 0.1, 0.01, 0.001, 0.001],
             }
```

```
Ajuste con Regresión Logística

Mejor parámetro:  {'C': 100}
Regresión logística train:  -0.050047331932499724
Regresión logística test:  -0.046329175837705766

Regresión logística train accuracy:  0.9891172178988327
Regresión logística test accuracy:  0.9892996108949417

Regresión logística train area bajo la curva:  0.9908136733631512
Regresión logística test area bajo la curva:  0.9919454563005188

              precision    recall  f1-score   support

  No ocupada       1.00      0.99      0.99      3185
     Ocupada       0.96      1.00      0.98       927

   micro avg       0.99      0.99      0.99      4112
   macro avg       0.98      0.99      0.98      4112
weighted avg       0.99      0.99      0.99      4112


Matriz de confusión:
 [[3144   41]
 [   3  924]]
```

Vemos que nuestro modelo tiende a evitar la regularización, un análisis de esto se dará más adelante.



Ajuste con Support Vector Machine

Siguiendo la recomendación, utilizamos un kernel RBF-Gaussiano.

También con CV ajustamos el valor de γ (para definir la similitud entre dos puntos) y C (para regularizar).

Cuando $\gamma = auto$, equivale a $\gamma = \frac{1}{nº\ características} \approx 0{,}07$

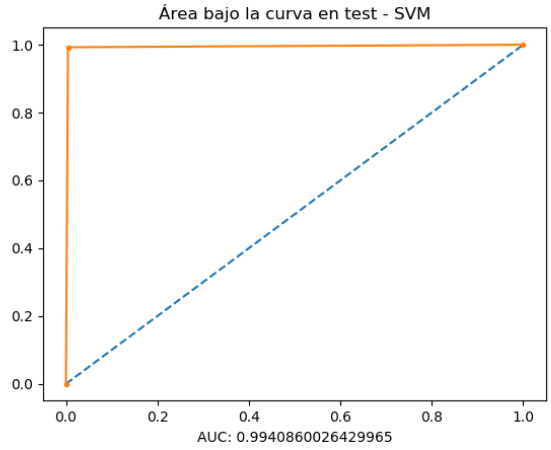
```
parameters = {
                'C': [1000, 100, 10, 1, 0.1, 0.001],
                'gamma': [0.001, 0.0001, 'auto']
              }
```

Al igual que con RF, esta búsqueda del mejor valor lleva demasiado tiempo y en el código los parámetros se reducen a los siguientes:

```
parameters = {
                'C': [100, 10],
                'gamma': [0.001, 'auto']
              }
```

Al tener un valor de γ alto vemos que se define una función Gaussiana con poca varianza, evitando que puntos distantes se consideren iguales.

```
Ajuste con SVM

Mejores parámetros:  {'C': 100, 'gamma': 'auto'}
Support Vector Machine train:  0.9962305447470817
Support Vector Machine test:  0.9922178988326849

Support Vector Machine train accuracy:  0.9962305447470817
Support Vector Machine test accuracy:  0.9922178988326849

Support Vector Machine train area bajo la curva:  0.996630558052519
Support Vector Machine test area bajo la curva:  0.9919582392303001

              precision    recall  f1-score   support

  No ocupada       1.00      0.99      0.99      3173
     Ocupada       0.97      0.99      0.98       939

   micro avg       0.99      0.99      0.99      4112
   macro avg       0.99      0.99      0.99      4112
weighted avg       0.99      0.99      0.99      4112


Matriz de confusión:
 [[3123   24]
 [   4  961]]
```



Ajuste con Boosting

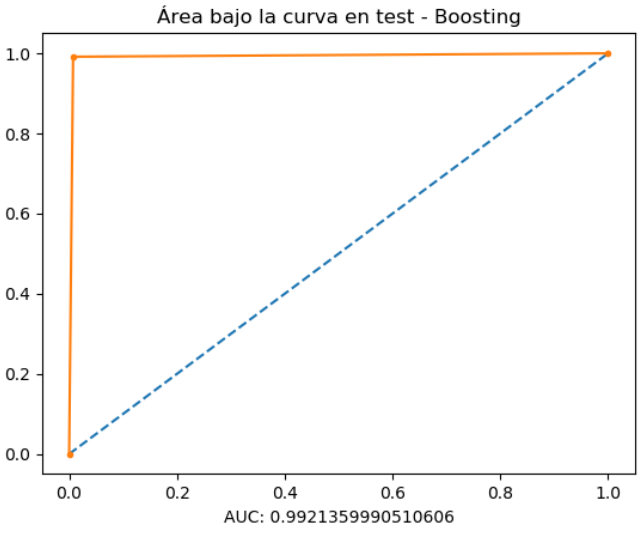
Se parte de lo sugerido, un algoritmo *AdaBoost* con funciones *stump*. Mediante *CV* se elige el mejor número de estimadores.

```
parameters = {
                'n_estimators': [50, 100, 200, 500, 1000]
              }
```

También es necesario reducir el tamaño de la búsqueda para la entrega, dejando el rango:

```
parameters = {
                'n_estimators': [50, 100]
              }
```

```
Ajuste con Boosting

Mejores parámetros:  {'n_estimators': 50}
AdaBoost train:  0.9910627431906615
AdaBoost test:  0.9910019455252919

AdaBoost train accuracy:  0.9910627431906615
AdaBoost test accuracy:  0.9910019455252919

AdaBoost train area bajo la curva:  0.9921600364532155
AdaBoost test area bajo la curva:  0.9923196912013535

              precision    recall  f1-score   support

  No ocupada       1.00      0.99      0.99      3152
     Ocupada       0.97      0.99      0.98       960

   micro avg       0.99      0.99      0.99      4112
   macro avg       0.98      0.99      0.99      4112
weighted avg       0.99      0.99      0.99      4112


Matriz de confusión:
 [[3120   32]
 [   5  955]]
```



Cálculos adicionales

Usando un modelo de regresión logística simple podemos medir qué precisión pueden alcanzar cada una de las características por separado, obteniendo:

```
Temperatura, train accuracy:  0.8134119649805448
Temperatura, test accuracy:  0.8207684824902723

Humedad relativa, train accuracy:  0.7703064202334631
Humedad relativa, test accuracy:  0.7636186770428015

Cantidad de lux, train accuracy:  0.9865029182879378
Cantidad de lux, test accuracy:  0.9868677042801557

CO2, train accuracy:  0.7843506809338522
CO2, test accuracy:  0.7862354085603113

Ratio de humedad, train accuracy:  0.778635700389105
Ratio de humedad, test accuracy:  0.769455252918288

Días-seno, train accuracy:  0.7703064202334631
Días-seno, test accuracy:  0.7636186770428015

Días-coseno, train accuracy:  0.7703064202334631
Días-coseno, test accuracy:  0.7636186770428015

Horas-seno, train accuracy:  0.7703064202334631
Horas-seno, test accuracy:  0.7636186770428015

Horas-coseno, train accuracy:  0.8120136186770428
Horas-coseno, test accuracy:  0.8081225680933852

Minutos-seno, train accuracy:  0.7703064202334631
Minutos-seno, test accuracy:  0.7636186770428015

Minutos-coseno, train accuracy:  0.7703064202334631
Minutos-coseno, test accuracy:  0.7636186770428015

Segundos-seno, train accuracy:  0.7703064202334631
Segundos-seno, test accuracy:  0.7636186770428015

Segundos-coseno, train accuracy:  0.7703064202334631
Segundos-coseno, test accuracy:  0.7636186770428015
```

Análisis de resultados y justificación del modelo

Después de analizar el efecto de cada variable en la clasificación vemos que únicamente con la luz podemos determinar la ocupación o no de la habitación de forma casi perfecta. Para entender esto, debemos fijarnos en las imágenes del principio y en la descripción sobre la creación de la muestra.

Es altamente probable que la habitación donde se realizaron los experimentos no contase con mucha iluminación natural. Por tanto, la propia iluminación artificial se habría encendido cada vez que alguien entrase en la sala, y simplemente teniendo en cuenta la cantidad de iluminación que hay se puede determinar la presencia.

Esta puede ser una de las razones por las que los modelos evitan la regularización. Al ser una muestra tan fácilmente clasificable en base a una única característica los modelos no tienden a aplicar un alto grado de regularización para obtener generalización.

Sobre nuestro modelo, según los resultados obtenidos, esto nos indica que hemos aprendido de forma muy buena a predecir si alguien ocuparía una habitación de esas características, pero un caso más general posiblemente se comporte de forma mucho peor.

Si quisiéramos que nuestro clasificador funcionase de forma más genérica, pero sin conseguir un conjunto de datos nuevos, una posible vía de aplicación sería la eliminación de la característica *luz* y volviendo a realizar el aprendizaje.

Bibliografía

Información sobre el conjunto de datos e ideas sobre cómo afrontar el problema:

1. https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+
2. https://www.sciencedirect.com/science/article/pii/S0378778815304357
3. https://github.com/LuisM78/Occupancy-detection-data
4. https://machinelearningmastery.com/how-to-predict-room-occupancy-based-on-environmental-factors/