OCUPACION DE UNA SALA DE OFICINA

June 20, 2019

José Antonio Garrido Sualdea

MACHINE LEARNING

Tema a tratar: "Detección precisa de la ocupación de una sala de oficina a partir de mediciones de luz, temperatura, humedad y CO2 utilizando modelos de aprendizaje estadístico. Luis M. Candanedo, Veronique Feldheim. Energía y edificios. Volumen 112 , 15 de enero de 2016, páginas 28-39. "

En este proyecto vamos a hacer uso de Machine learning supervisado, es decir, tenemos una etiqueta de la cual podemos aprender. El conjunto de datos provisto tiene 6 variables independientes (predictores). La ocupación es una variable categórica con 2 niveles: 0 para no ocupada; y 1 por ocupado.

Del conjunto de variables independientes, he decidido quitar la referente al tiempo debido a que no tiene porque precisar bien del todo la ocupación de una sala de oficina. Por ejemplo, la organización una fiesta el fin de semana o que justo algún dia festivo pueda coincidir con un dia laboral. Por lo tanto esos datos no son adecuados como conjunto de datos de entrenamiento.

```
In [3]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import scipy.optimize as opt

        from sklearn.svm import SVC
```

DATOS INFORMACIÓN

fecha hora año-mes-día hora: minuto: segundo

Temperatura, en Celsius

Humedad relativa,%

Luz, en Lux

CO2, en ppm

Relación de humedad, Cantidad derivada de la temperatura y humedad relativa, en kg Agua-vapor / kg-aire

Ocupación, 0 o 1, 0 para no ocupado, 1 para estado ocupado

DATOS DE ENTRENAMIENTO

```
In [4]: columnNames = ['Temperature', 'Humidity', 'Light', 'CO2', 'HumidityRatio']
        dataTraining = pd.read_csv('datatraining.txt')
        dataTraining.columns = dataTraining.columns.str.strip()
        X = dataTraining[columnNames].values
        Y = dataTraining['Occupancy'].values
```

```
        (m, n) = X.shape
        X = np.c_[X, np.ones(m)]
```

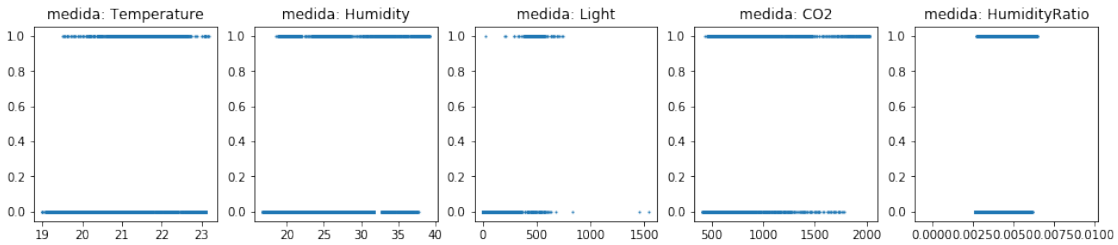
```
In [5]: dataTraining.head()
```

Out[5]:

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
1	2015-02-04 17:51:00	23.18	27.2720	426.0	721.25	0.004793	1
2	2015-02-04 17:51:59	23.15	27.2675	429.5	714.00	0.004783	1
3	2015-02-04 17:53:00	23.15	27.2450	426.0	713.50	0.004779	1
4	2015-02-04 17:54:00	23.15	27.2000	426.0	708.25	0.004772	1
5	2015-02-04 17:55:00	23.10	27.2000	426.0	704.50	0.004757	1

```
In [6]: Temperature = dataTraining[['Temperature']].values
        Humidity = dataTraining[['Humidity']].values
        Light = dataTraining[['Light']].values
        CO2 = dataTraining[['CO2']].values
        HumidityRatio = dataTraining[['HumidityRatio']].values
        vector = [Temperature, Humidity, Light, CO2, HumidityRatio]
        vectorName = columnNames

        fig, axs = plt.subplots(1,5,figsize=(16,3))
        (m, n) = dataTraining.shape
        for ax, v, vn in zip(axs, vector, vectorName):
            ax.scatter(v, Y, s=1)
            ax.set_title(' medida: {} '.format(vn))
```

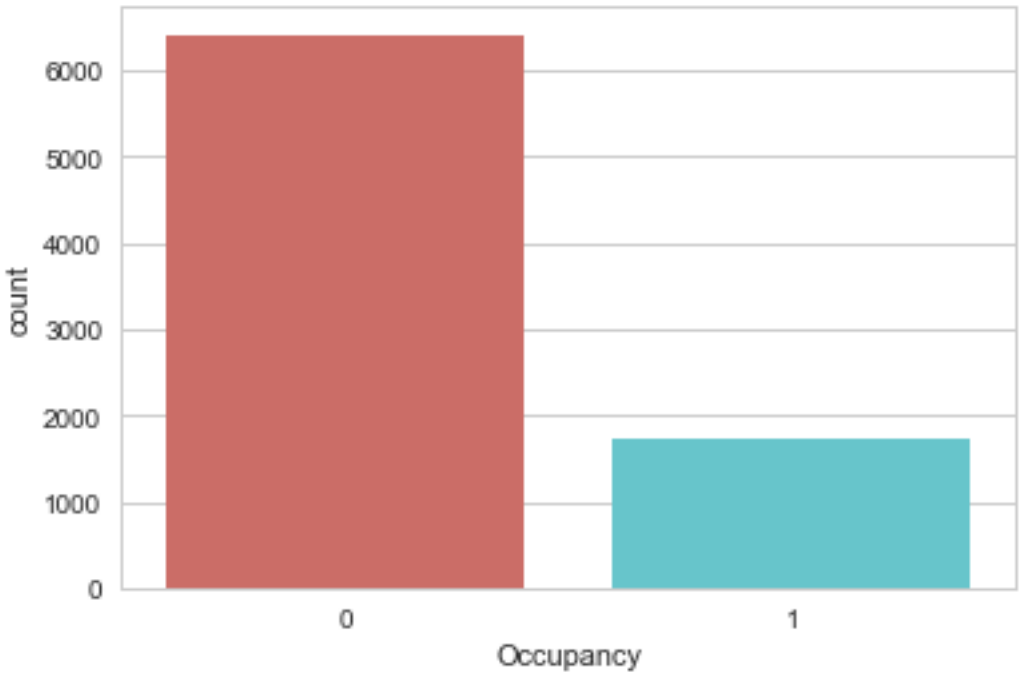


Distribución del dataset

```
In [7]: dataTraining['Occupancy'].value_counts()
```

```
Out[7]: 0    6414
        1    1729
        Name: Occupancy, dtype: int64
```

```
In [8]: import seaborn as sns
        sns.set(style="white")
        sns.set(style="whitegrid", color_codes=True)
        sns.countplot(x='Occupancy', data=dataTraining, palette='hls')
        plt.show()
```



DATOS DE VALIDACIÓN

```
In [9]: dataValidation = pd.read_csv('datatest.txt')
        dataValidation.columns = dataValidation.columns.str.strip()
        X_1 = dataValidation[columnNames].values
        Y_1 = dataValidation['Occupancy'].values
        (m_1, n_1) = X_1.shape
        X_1 = np.c_[X_1, np.ones(m_1)]

In [10]: dataValidation.head()
```

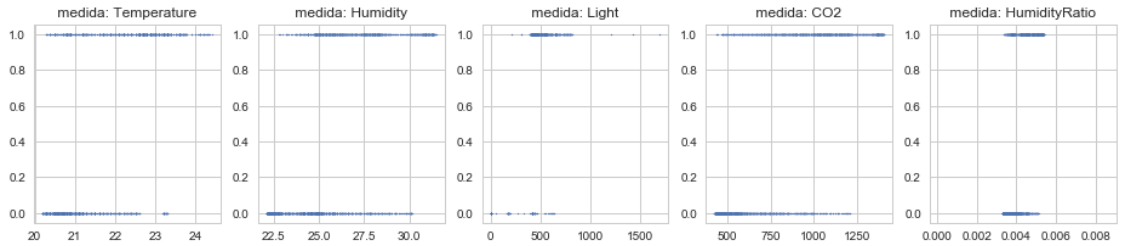
Out[10]:

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
140	2015-02-02 14:19:00	23.7000	26.272	585.200000	749.200000	0.004764	1
141	2015-02-02 14:19:59	23.7180	26.290	578.400000	760.400000		
142	2015-02-02 14:21:00	23.7300	26.230	572.666667	769.666667		
143	2015-02-02 14:22:00	23.7225	26.125	493.750000	774.750000		
144	2015-02-02 14:23:00	23.7540	26.200	488.600000	779.000000		

```
141       0.004773          1
142       0.004765          1
143       0.004744          1
144       0.004767          1
```

```
In [11]: Temperature = dataValidation[['Temperature']].values
         Humidity = dataValidation[['Humidity']].values
         Light = dataValidation[['Light']].values
         CO2 = dataValidation[['CO2']].values
         HumidityRatio = dataValidation[['HumidityRatio']].values
         vector = [Temperature, Humidity, Light, CO2, HumidityRatio]
         vectorName = columnNames

         fig, axs = plt.subplots(1,5,figsize=(16,3))
         (m, n) = dataValidation.shape
         for ax, v, vn in zip(axs, vector, vectorName):
             ax.scatter(v, Y_1, s=1)
             ax.set_title(' medida: {} '.format(vn))
```



DATOS PARA TEST

```
In [12]: dataTest = pd.read_csv('datatest2.txt')
         dataTest.columns = dataTest.columns.str.strip()
         X_2 = dataTest[columnNames].values
         Y_2 = dataTest['Occupancy'].values
         (m_2, n_2) = X_2.shape
         X_2 = np.c_[X_2, np.ones(m_2)]
```

```
In [13]: dataTest.head()
```

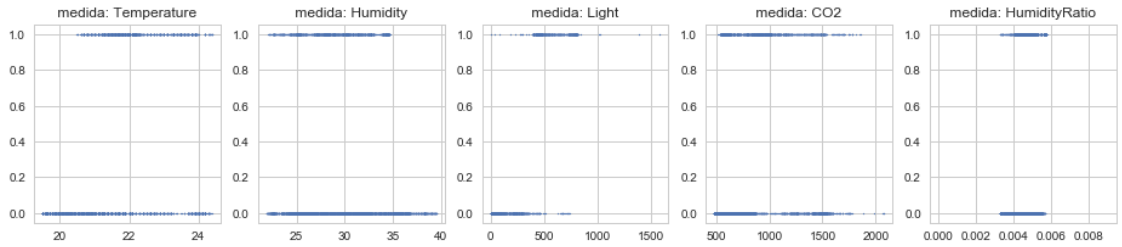
```
Out[13]:                  date  Temperature   Humidity       Light          CO2  \
         1  2015-02-11 14:48:00      21.7600  31.133333  437.333333  1029.666667
         2  2015-02-11 14:49:00      21.7900  31.000000  437.333333  1000.000000
         3  2015-02-11 14:50:00      21.7675  31.122500  434.000000  1003.750000
         4  2015-02-11 14:51:00      21.7675  31.122500  439.000000  1009.500000
         5  2015-02-11 14:51:59      21.7900  31.133333  437.333333  1005.666667

            HumidityRatio  Occupancy
         1       0.005021          1
```

```
2       0.005009          1
3       0.005022          1
4       0.005022          1
5       0.005030          1
```

```
In [14]: Temperature = dataTest[['Temperature']].values
         Humidity = dataTest[['Humidity']].values
         Light = dataTest[['Light']].values
         CO2 = dataTest[['CO2']].values
         HumidityRatio = dataTest[['HumidityRatio']].values
         vector = [Temperature, Humidity, Light, CO2, HumidityRatio]
         vectorName = columnNames

         fig, axs = plt.subplots(1,5,figsize=(16,3))
         (m, n) = dataValidation.shape
         for ax, v, vn in zip(axs, vector, vectorName):
             ax.scatter(v, Y_2, s=1)
             ax.set_title(' medida: {} '.format(vn))
```



.....

TÉCNICA: SUPPORT VECTOR MACHINES(SVM)

.....

KERNEL LINEAR con penalizacion 1

```
In [ ]: #Creo un modelo SVC (support vector clasification) con el kernel lineal
        #y el parametro C (penalización) igual a 1
        svm = SVC(kernel='linear', C=1)
        svm.fit(X, Y)

        #Puntuación sobre los distintos conjuntos de datos
        print('Training score: {}'.format(svm.score(X, Y)))
        print('Validation score: {}'.format(svm.score(X_1, Y_1)))
        print('Test score: {}'.format(svm.score(X_2, Y_2)))
```

```
Training score: 0.9862458553358713
Validation score: 0.9786116322701689
Test score: 0.992616899097621
```

KERNEL LINEAR con penalizacion 100

```
In [29]: #Creo un modelo SVC (support vector clasification) con el kernel lineal
         #y el parametro C (penalización) igual a 100
         svm = SVC(kernel='linear', C=100)
         svm.fit(X, Y)

         #Puntuación sobre los distintos conjuntos de datos
         print('Training score: {}'.format(svm.score(X, Y)))
         print('Validation score: {}'.format(svm.score(X_1, Y_1)))
         print('Test score: {}'.format(svm.score(X_2, Y_2)))
```

```
Training score: 0.9868598796512342
Validation score: 0.9786116322701689
Test score: 0.9928219852337982
```

KERNEL GAUSSIANO

Es posible que durante el entrenamiento el modelo sufra subreajuste(el modelo se aprende los datos de entrenamiento muy bien pero los datos de validación no son exactamente igual) El modelo aprendido en el entrenamiento no funciona muy bien en los datos de test y no conseguimos un resultado tan robusto como en la regresión lineal por consiguiente.

```
In [17]: #Creo un modelo SVC (support vector clasification) con el kernel rbf
         #y realizamos una búsqueda de los mejores parámetros de configuración
         max_score = 0
         best_C = None
         best_sigma = None
         best_model = None
         for C in [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]:
             for sigma in [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]:
                 svm_1 = SVC(kernel='rbf', C=C, gamma=1/(2*sigma**2))
                 svm_1.fit(X, Y)
                 score = svm_1.score(X_1, Y_1)
                 if score > max_score:
                     max_score = score
                     best_C = C
                     best_sigma = sigma
                     best_model = svm_1
         print('Mejor modelo: C: {}, sigma: {}'.format(best_C, best_sigma))
         #Puntuación sobre los distintos conjuntos de datos
         print('Training score: {}'.format(best_model.score(X, Y)))
         print('Validation score: {}'.format(best_model.score(X_1, Y_1)))
         print('Test score: {}'.format(best_model.score(X_2, Y_2)))
```

```
Mejor modelo: C: 0.03, sigma: 30
Training score: 0.9848950018420729
Validation score: 0.9703564727954972
Test score: 0.8964315012305168
```

.....

REGRESION LOGISTICA REGULARIZADA

.....

Función sigmoide, para interpretar la salida como una probabilidad

```
In [ ]: def g(z):
            return 1/(1 + np.exp(-z))
```

Función de coste

```
In [ ]: def coste(T, X, Y, m, l):
            return -(1/m)*(
                (np.log(g(X.dot(T))).transpose().dot(Y) +
                (np.log(1-g(X.dot(T)))).transpose().dot(1-Y))
            ) + (l/(2*m))*np.sum(T**2)
```

Función gradiente

```
In [ ]: def gradiente(T, X, Y, m, l):
            T_ = T.copy()
            T_[0] = 0
            return (1/m)*X.transpose().dot((g(X.dot(T))-Y)) + (l/m)*T_
```

Cálculo del valor óptimo de los parámetros

```
In [ ]: #para obtener el valor de los parámetros theta que minimizan
        #la función de coste para la regresión logística
        l = 1
        #sampling, obtiene datos de manera aleatoria. Inizializamos la semilla a 0
        T = np.array([0]*n)
        result = opt.fmin_tnc(func=coste, x0=np.array([0]*6), fprime=gradiente, args=(X, Y, m,
        theta_opt = result[0]
```

Evaluación de la regresión logística con dataTraining

```
In [ ]: n_aciertos = 0
        for i in range(m):
            prediccion = 0
            if g(X[i].dot(theta_opt)) > 0.5:
                prediccion = 1
            if prediccion == Y[i]:
                n_aciertos += 1

        print('Training score: {} %'.format(100 * n_aciertos / m))
```

```
Training score: 96.84803001876173 %
```

Evaluación de la regresión logística con dataValidation

```
In [ ]: n_aciertos = 0
        for i in range(m):
            prediccion = 0
            if g(X_1[i].dot(theta_opt)) > 0.5:
                prediccion = 1
            if prediccion == Y_1[i]:
                n_aciertos += 1

        print('Validation score: {} %'.format(100 * n_aciertos / m))
```

```
Validation score: 97.82363977485929 %
```

Evaluación de la regresión logística con dataTest

```
In [ ]: n_aciertos = 0
        for i in range(m):
            prediccion = 0
            if g(X_2[i].dot(theta_opt)) > 0.5:
                prediccion = 1
            if prediccion == Y_2[i]:
                n_aciertos += 1

        print('Test score: {} %'.format(100 * n_aciertos / m))
```

```
Test score: 98.98686679174484 %
```

Vemos como a medida que entrenamos el algoritmo, este es capaz de predecir mejor, consiguiendo un porcentaje del 98.98% en los datos de Test

.

PESOS DE LAS COLUMNAS

Theta_opt representa los pesos de las columnas en el modelo de regresión logística que minimizan el error. Es decir, son los mejores pesos que se han encontrado para cada columna. En este caso ha salido resultante la temperatura con un valor del 0,6. El hecho de que sea negativo quiere decir que hay una relacción inversamente proporcional.

```
In [28]: for T, v in zip(theta_opt, columnNames):
             print('columna: {}, peso: {}'.format(v, T))
```

```
columna: Temperature, peso: -0.6129582328491153
columna: Humidity, peso: 0.03368066330724766
columna: Light, peso: 0.019931379278129886
columna: CO2, peso: 0.004511695911580915
columna: HumidityRatio, peso: -0.013553500706577341
```