Seminario PYTHON - MCAF (Sesión III)

Francisco Gárate Santiago

UCM, 2022

Objetivos de la sesión

Probabilidad y estadística en Python:

- Introducción a la librería estadística SciPy.stats
- Manejo de las distribuciones de probabilidad más conocidas
- Modelos estadísticos.

Introducción al Machine Learning y a los principales algoritmos estadísticos de aprendizaje automático:

- Familiarizarse y conocer la librería de aprendizaje automático Scikit-learn.
- Conocer el análisis de componentes y estimación de un valor con `scikit-learn.LinearRegression`
- Introducción a la metodologia básica del machine learning (model fitting, predicting, cross-validation, etc.)
- Overfitting y Cross-Validation: Técnica de entrenamiento y validación.

Scipy

stats

La librería **SciPy** (*Scientific computing*, pronunciado "Sigh Pie") posee diferentes módulos (**stats, linalg, interpolate...**) orientados a diversas áreas científicas. De momento, el módulo/paquete que nos interesa para estadística es **scipy.stats**.

- Sitio: `https://www.scipy.org/scipylib/index.html`
- Documentacion: `https://docs.scipy.org/doc/scipy/reference/tutorial/stats.html`
- Licencia open-source propia.

Scipy

scipy.stats

- **scipy.stats** aporta algunas funciones que pueden complementar a NumPy:

```
1 import numpy as np
2 import scipy.stats as st
3 np.random.seed(999)
4 x = np.random.normal(size=1000)
5 print(np.median(x))
6 print(st.scoreatpercentile(x, 50)) # mediana
7 print(st.scoreatpercentile(x, 95))
```

```
0.007576375024717093
0.007576375024717093
1.6906947688452745
```

Distribuciones de probabilidad

scipy.stats

Sin embargo, donde destaca **SciPy** es en el manejo de **distribuciones de probabilidad**:

- En estadística, toda variable aleatoria se distribuye en base a una función de probabilidad.
- Dependiendo del evento aleatorio, las distribuciones pueden ser **discretas** (toma ciertos valores aislados -puntos- dentro de un intervalo) o **continuas** (puede tomar cualquier valor real del rango).

Distribuciones de probabilidad

scipy.stats

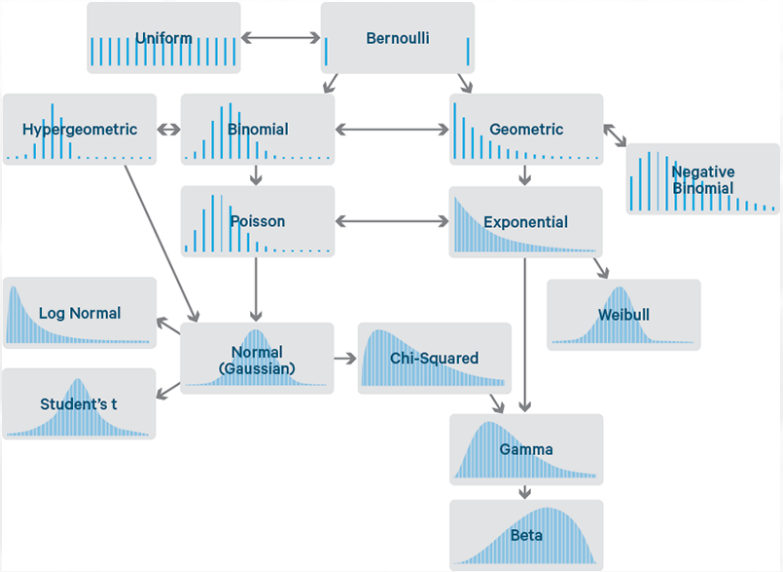
La versión 1.6 de SciPy tiene un catálogo de 88 distribuciones continuas y 15 discretas:

```
1 import scipy.stats as st
2 print(dir(st))
```

- **.norm()**: Normal (Gausiana)
- **.lognorm()**: Lognormal
- **.expon()**: Exponencial
- **.t()**: Student's t
- **.chi2()**: Chi-cuadrado
- **.gamma()**: Gamma
- **.beta()**: Beta
- **.bernoulli()**: Bernoulli
- **.binom()**: Binomial
- **.poisson()**: Poisson

`https://docs.scipy.org/doc/scipy/reference/tutorial/stats/continuous.html`

Distribuciones de probabilidad



Distribuciones de probabilidad

scipy.stats

Función de densidad:

- Cada distribución puede ilustrarse con su **función de densidad de probabilidad** (pdf), es decir, el eje horizontal es el conjunto de posibles resultados numéricos y el eje vertical describe la probabilidad de resultados.
- En la función de densidad de probabilidad, la **suma** de las alturas de las líneas (distrib. discretas) o las áreas bajo las curvas (distrib. continuas) son **siempre 1**.

Distribuciones de probabilidad

scipy.stats

Tomaremos como ejemplo la distribución normal (gaussina) como la función de distribución de nuestra variable a estudiar. La distr. normal es ampliamente usada en inferencia estadística, ya que con muestras grandes los errores *se aproximan bien* a una distribución normal.
La **función de densidad de probabilidad** normal para un número real x es:

$$f(x) = \frac{exp(-x^2/2)}{\sqrt{2\pi}}$$

norm.pdf(x, loc=0, scale=1) para el valor x, siendo:

- **loc**: media de la distribución (por defecto 0)
- **scale**: desviación estándar de la distribución (por defecto 1).

```
1 import scipy.stats as st
2 a = st.norm.pdf(0.5)
3 print(a)
```

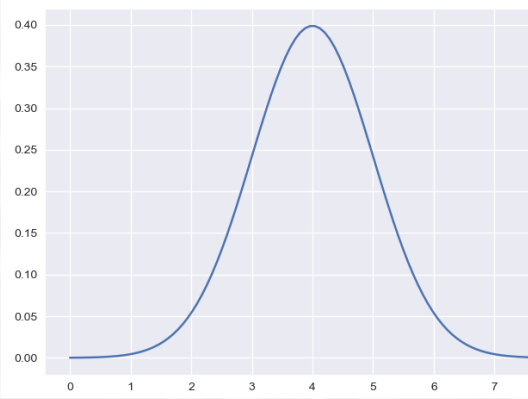
```
0.3520653267642995
```

Distribuciones de probabilidad

scipy.stats

Ejemplo usando ***nombre_distribución*.pdf(x)** con media 4 y desviación estándar 1:

```
1  import numpy as np
2  import scipy.stats as st
3  import matplotlib.pyplot as plt
4  gaussian = st.norm(loc=4.0, scale=1.0)
5  x = np.linspace(0.0, 8.0, 100)
6  y = gaussian.pdf(x)
7  plt.style.use('seaborn')
8  plt.plot(x, y)
9  plt.show()
```



Distribuciones de probabilidad

scipy.stats

Adicionalmente, disponemos de la función de distribución acumulativa y la función de punto porcentual (la inversa de cdf):

- **.cdf(x, loc=0, scale=1)**: Función de distribución acumulativa (*Cumulative distribution function*)
- **.ppf(q, loc=0, scale=1)**: Función de punto porcentual (*Percent point function*) (la inversa de cdf, equivalente a los percentiles).

```
1 import scipy.stats as st
2 gaussian = st.norm(loc=4.0, scale=1.0)
3 a = st.norm.cdf(1.65)
4 b = st.norm.ppf(0.95) #percentil
5 print(a, b)
```

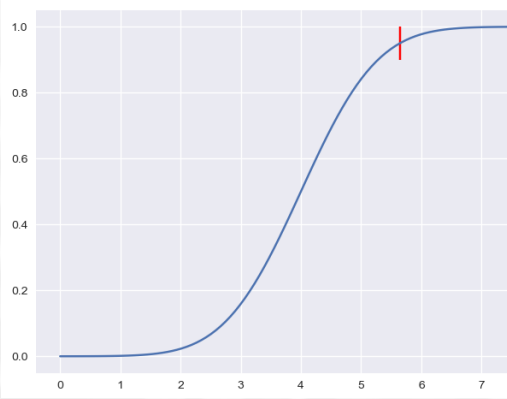
```
0.9505285319663519
1.6448536269514722
```

Distribuciones de probabilidad

scipy.stats

Ejemplo usando **.cdf(x)** y **.ppf(0.95)** para calcular el percentil 95

```
1  import numpy as np
2  import scipy.stats as st
3  import matplotlib.pyplot as plt
4  gaussian = st.norm(loc=4.0, scale=1.0)
5  x = np.linspace(0.0, 8.0, 100)
6  y = gaussian.cdf(x)
7  plt.style.use('seaborn')
8  plt.plot(x, y)
9  plt.vlines(gaussian.ppf(0.95),0.9,1,'r')
10 plt.show()
```

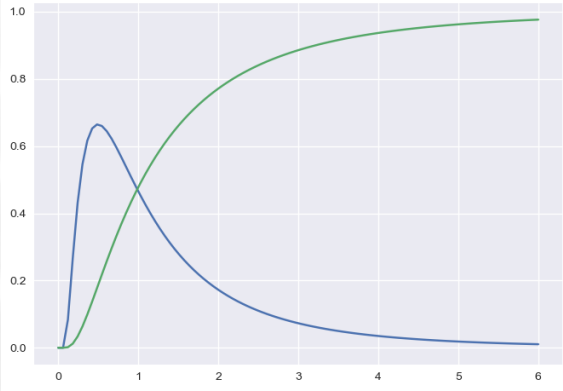


Distribuciones de probabilidad

Ejemplo con la función de lognormalidad con **st.lognorm(s=sigma, scale=exp(mu))**

s = sigma std, mu = mean
sigma = std / la media de ln(x)
std = desviación estandar de ln(x)

```
1  import numpy as np
2  import scipy.stats as st
3  import matplotlib.pyplot as plt
4  # std normal
5  sigma = 0.859455801705594
6  # mean normal
7  mu = 0.418749176686875
8  lognormal = st.lognorm(s=sigma,
       scale=np.exp(mu))
9  x = np.linspace(0, 8.0, 100)
10 plt.style.use('seaborn')
11 plt.plot(x, lognormal.pdf(x))
12 plt.plot(x, lognormal.cdf(x))
13 plt.show()
```

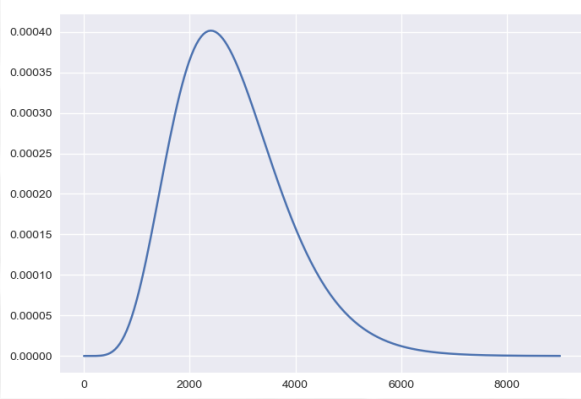


Distribuciones de probabilidad

st.gamma(a=alpha, scale=1/lambda)
a: alpha
scale: theta = 1 / lambda (*por defecto = 1*)
st.gamma.pdf(x, a, scale)
x: valor para evaluar la función

Cuando a es un entero, gamma se reduce a la distribución Erlang, y cuando a=1 a la distribución exponencial.

```
1  import numpy as np
2  import scipy.stats as st
3  from matplotlib import pyplot as plt
4
5  alpha = 7.
6  lamdba = 0.0025
7  x = np.linspace (0, 9000, 200)
8  y = st.gamma.pdf(x, a=alpha, scale
       =1/lamdba)
9
10 plt.style.use('seaborn')
11 plt.plot(x, y)
12 plt.show()
```



Distribuciones de probabilidad

scipy.stats

***mi_distribución*.stats(loc=0, scale=1, moments='mvsk')**: Devuelve los momentos que especifiquemos:

- **m:** Media *(mean)*
- **v:** Varianza *(variance)*
- **s:** Sesgo *(skew)*
- **k:** kurtosis

```
1 import scipy.stats as st
2 gaussian = st.norm(loc=4.0, scale=1.0) #mu=4 #sigma=1
3 mean, var, skew, kurt = gaussian.stats(moments='mvsk')
4 print(mean, var, skew, kurt)
```

```
4.0   1.0   0.0   0.0
```

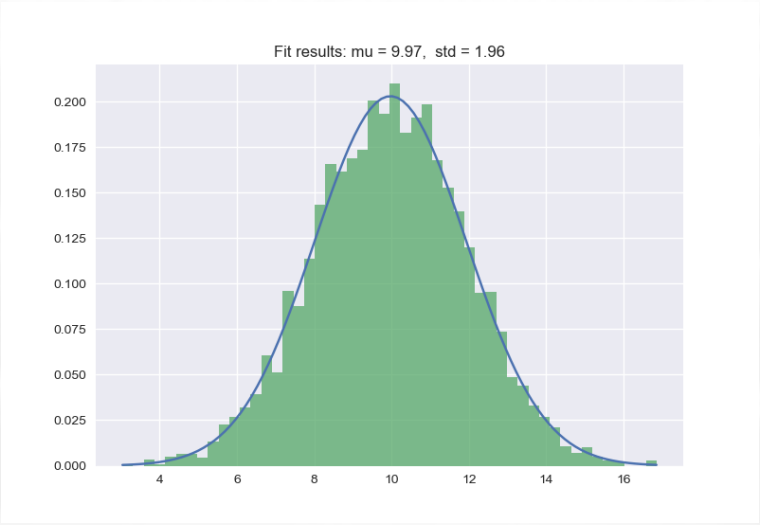
Modelos de Ajuste a una distribución

SciPy.Stats

- Para ajustar datos a una distribución, es decir, determinar la distribución que sigue un vector de datos, todas las distribuciones continuas de **SciPy.Stats** disponen la función **.fit()**, que nos permite estimar parámetros partiendo de los datos disponibles.
- El método utilizado es el de máxima verosimilitud.
- El output de fit dependerá de los parámetros de la distribución a ajustar. Por ejemplo, en las siguientes dos distribuciones el output es distinto:
  - Normal: **st.norm.fit(data)**, devuelve media y std
  - Lognormal: **st.lognormal.fit(data)**, devuelve sigma, mu y std.
- Importante: **.fit()** no da ningún valor sobre la **bondad del ajuste**. Para ello veremos otras funciones.

Modelos de Ajuste a una distribución

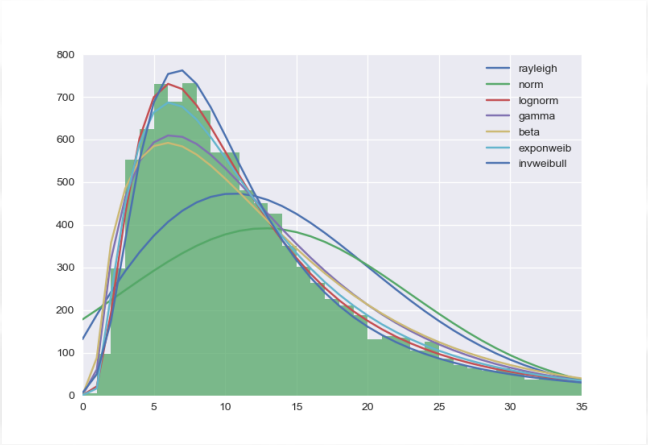
SciPy.Stats + matplotlib



Modelos de Ajuste a una distribución

SciPy.Stats + matplotlib

Para un mismo set de datos puede realizarse para tantas distribuciones como queramos, y elegir la que mejor se adapte:



Modelos de Ajuste a una distribución

SciPy.Stats

El código del anterior gráfico está disponible en el fichero **Ejemplo_Ajuste_distribucion.ipynb**. A continuación, un extracto del mismo:

```
size = 10000
x = np.arange(size)
y = np.random.lognormal(np.log(10), np.log(2), size)

distrs = ['rayleigh', 'norm', 'lognorm', 'gamma', 'beta', 'exponweib', '
    invweibull']
for i in distrs:
    dist = getattr(st, i)
    param = dist.fit(y)
    pdf_fitted = dist.pdf(x, *param[:-2], loc=param[-2], scale=param[-1])
        * size
    plt.plot(pdf_fitted, label=i)
    plt.xlim(0,25)
plt.hist(y, bins=range(50), alpha=0.75)
plt.show()
```

Modelos estadísticos

Modelos de ajustes a los datos

Muchos procedimientos estadísticos suponen que los datos siguen algún tipo de **modelo matemático que se define mediante una ecuación**, donde al desconocerse alguno de sus parámetros, pueden estimarse a partir de la información disponible.

Existen principalmente dos procedimientos para estimar los coeficientes de un modelo de regresión, o para estimar los parámetros de una distribución de probabilidad:

- **Mínimos cuadrados ordinarios (MCO)**, *ordinary least squares* (OLS) o mínimos cuadrados lineales son los nombres que recibe el método para encontrar los parámetros poblacionales en un modelo de regresión lineal obtenidos minimizando los residuos observados (la suma de los cuadrados de las diferencias entre la variable dependiente observada).
- **Estimación por máxima verosimilitud (EMV)**, *maximum likelihood estimation* (MLE), la cual requiere maximizar la función de verosimilitud, o función de probabilidad conjunta de los valores muestrales.

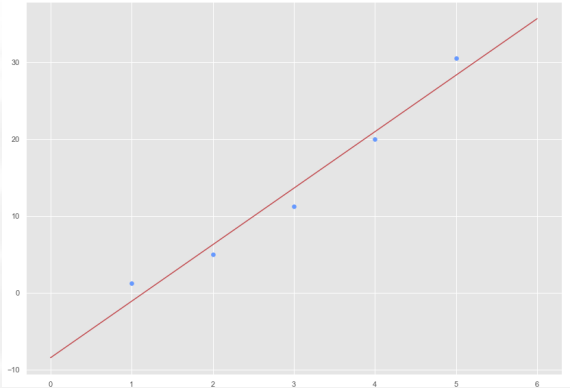
Bajo la hipótesis de distribución normal, las estimaciones por ambos criterios coinciden.

Regresión lineal

Mínimos cuadrados ordinarios

Gráfico de puntos y recta estimada de regresión por mínimos cuadráticos:

	x	y
2014	1	1.25
2015	2	5
2016	3	11.25
2017	4	20
2018	5	30.5




```
df = pd.DataFrame(index=range(2014,2019))
df['x'] = [1, 2, 3, 4, 5]
df['y'] = [1.25, 5, 11.25, 20, 30.5]
```

Regresión lineal

Modelos de ajustes a los datos

En el modelo de regresión lineal, conocido también como modelo de regresión simple, o recta poblacional, es un modelo matemático usado para aproximar la relación de dependencia entre una variable dependiente y, las variables independientes x y un término aleatorio e:

$$y = \beta_1 + \beta_2 x + e$$

Siendo:

- **y**: la variable endógena o dependiente (variable explicada o regresando)
- **x**: la variable exógena o independiente (variable explicativa, regresor, covariable o variable de control)
- **e**: el error o perturbación aleatoria. Lo que no explica el modelo de ajuste.
- β_1: es la ordenada al origen (indica el valor de y cuando x=0).
- β_2 es la pendiente de la recta (cuanto cambia y por cada incremento de x).

El objetivo principal del modelo de regresión es β_1 y β_2 (parámetros fijos y desconocidos) a partir de una muestra dada.

Regresión lineal

Mínimos cuadrados ordinarios

En base a una muestra de tamaño n es posible estimar los parámetros del modelo.
Si una recta pasa exactamente por los puntos, el error será cero. El programa de minimización es el siguiente:

$$min \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_i))^2$$

En Python existen funciones específicas cuya finalidad es ajustar los datos tanto a una recta como a una distribución de probabilidad.

Así, para determinar la distribución que sigue un set de datos contamos en Python principalmente con dos librerías:

- Scipy: **scipy.stats**
- Statsmodels: **statsmodels.api**

Regresión lineal

SciPy.stats

SciPy.stats.linregress(x, y) donde la salida devuelve los siguientes parámetros:

- **slope**: pendiente de la línea de regresión
- **intercept**: intercepción de la línea de regresión
- **r_value**: coeficiente de correlación
- **p_value**: p-value cuya hipótesis nula es que la pendiente es cero.
- **std_err**: Error estándar de la estimación

```
1 import scipy.stats as st
2 x = (1., 2., 3., 4., 5.)
3 y = (1.25, 5, 11.25, 20, 30.5)
4 print(st.linregress(x,y))
```

```
LinregressResult(slope=7.35, intercept=-8.45, rvalue=0.983437109,
pvalue=0.00255244, stderr=0.782091213)
```

Statsmodels

Por último, la librería **statsmodels** proporciona funciones para la estimación de muchos modelos estadísticos diferentes, así como para la realización de pruebas estadísticas y la exploración de datos estadísticos.

- Statsmodels *(statistical models)*
- **Principales utilidades**: Estimación de diferentes modelos estadísticos, realización de pruebas estadísticas y exploración de datos.
- **Sitio**: `https://www.statsmodels.org/`
- **Documentación**: `www.statsmodels.org/stable/index.html#basic-documentation`
- **Licencia**: open-source BSD-Modified

Regresiones multiples

statsmodels.formula.api

stastmodels posee la clase **statsmodels.formula.api** con la función ols, la cual puede ser utilizada para ajustar regresiones lineales múltiples:

- **ols(formula, datos).fit()**

Ejemplo de estimación con regresión múltiple de la tasa de empleo en Estados Unidos utilizando el dataset "*U.S. economic time series used by Schotman & Dijk, 1988*".

```
1  from statsmodels.formula.api import ols
2  import pandas as pd
3
4  datos = pd.read_csv('desempleo.csv', sep=';', decimal=',')
5  formula = 'TOTEMP ~ GNPDEFL + GNP + UNEMP + ARMED + POP + YEAR'
6  results = ols(formula, datos).fit()
7  hipotesis = 'GNPDEFL = GNP, UNEMP = 2, YEAR/1829 = 1'
8  t_test = results.t_test(hipotesis)
9  print(t_test)
```

Modelo lineal generalizado (GLM)

statsmodels.formula.api.glm

- Scikit learn tiene una función llamada Generalized Linear Model, aunque es simplemente un modelo lineal, sin función de enlace, por lo que propiamente no es un GLM.
- En cambio, **Statsmodel** si dispone de una función GLM:

```
1  import pandas as pd
2  import statsmodels.api as sm
3  import statsmodels.formula.api as smf
4
5  data = pd.read_csv('airlines.csv')
6  formula='pf ~ cost + output + lf'
7
8  model = smf.glm(formula=formula, data=data, family=sm.families.Poisson())
9  result = model.fit()
10 print(result.summary())
```

Modelo lineal generalizado (GLM)

statsmodels.formula.api.glm

- A continuación se muestran las diferentes funciones de enlace posibles dependiendo de la familia de funciones:

	log	logit	probit	nbinom
Binomial	x	x	x	
Gamma	x			
Gaussian	x			
InverseGaussian	x			
NegativeBinomial	x			x
Poisson	x			
Tweedie	x			

Modelo lineal generalizado (GLM)

statsmodels.formula.api.glm

- Cuando nuestro GLM utilice la función Binomial, caso de modelización de variables de tipo binario, donde la variable puede tomar únicamente dos posibles valores 0 (fracaso) y 1 (éxito)
- Si se omite la función formula, debe indicarse: `glm_binom = smf.glm(data.endog, data.exog, family=sm.families.Binomial())`. Ejemplo:

```
1 import pandas as pd
2 import statsmodels.api as sm
3 import statsmodels.formula.api as smf
4
5 model = smf.glm(data.endog, data.exog, family=sm.families.Binomial(sm.
      families.links.logit()))
6 result = model.fit()
7 print(result.summary())
```

Otros ejemplos:

`https://www.statsmodels.org/dev/examples/notebooks/generated/glm.html`

Ejercicio 5

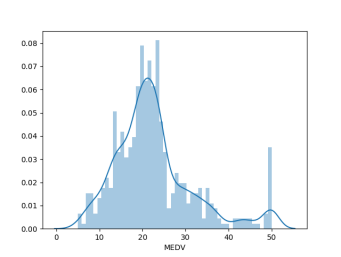
Ejercicio 5. Regresión lineal y GLM - Precio vivienda en Boston

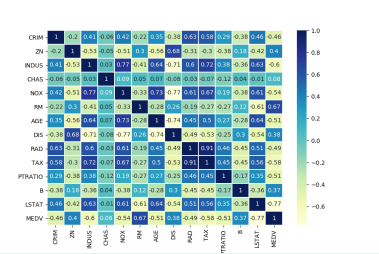
Veamos una aproximación al aprendizaje automático con el set de datos de Boston (boston.xlsx), en un primer lugar bajo un enfoque de modelos predictivos clásicos (regresión lineal y GLM, sin datos de entrenamiento ni test):

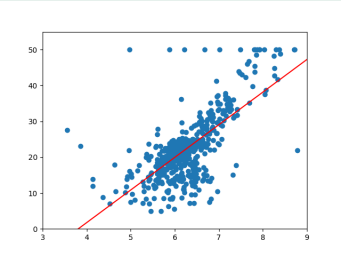
- Mostrar los datos a estimar (precio de la vivienda) a través de un histograma.
- Visualización de las correlaciones entre las variables.
- Realizar un análisis visual uni-variante
- Estimar el precio de la vivienda usando:
  - un modelo de regresión lineal con stasmodels: Hallar la recta de regresión entre la variable número de habitación (RM) y el valor de las viviendas (MEDV).
  - un modelo GLM con stasmodels utilizando el resto de variables cuantitativas.

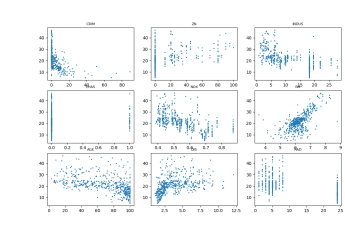
Ejercicio 5

Gráficos:









Ejercicio 5

Resultados

- Comparamos los resultados:

Función de regresión	RMSE
Regresión lineal	7.17
GLM	5.99

- Para validar dichas regresiones es útil la **validación cruzada**, es decir, entrenar el modelo con p.ej. el 75% de los datos, y ver cómo de fiable es estimando el 25% restante (muestra de contraste), técnica conocida como Leave-One-Out (Dejar uno fuera) o Cross-Validation.

Scikit-learn

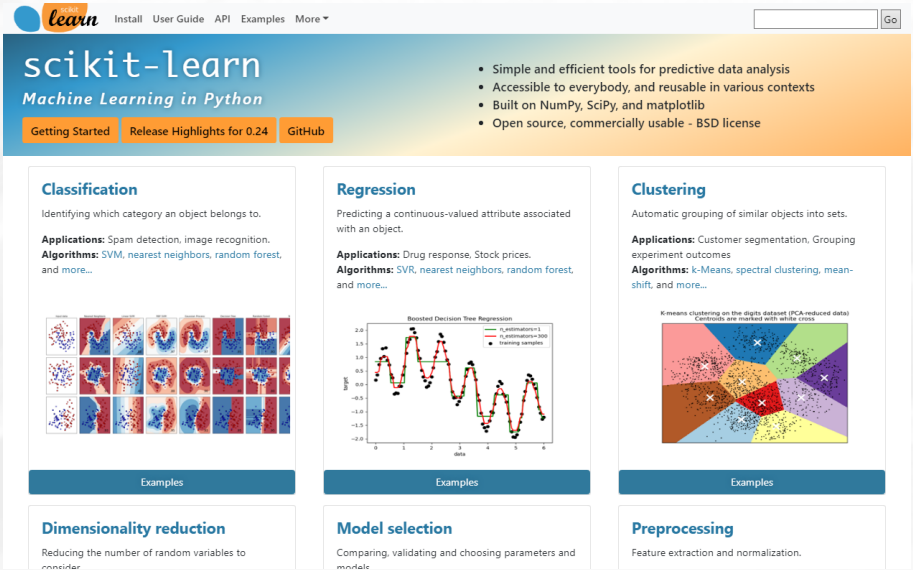
Machine learning

Scikit-learn es una biblioteca de aprendizaje automático (machine learning) de código abierto que soporta el aprendizaje supervisado y no supervisado. También proporciona varias herramientas para el ajuste de modelos, el preprocesamiento de datos, la selección y evaluación de modelos y muchas otras utilidades.

- Sitio: `https://scikit-learn.org/`
- Documentacion: `https://scikit-learn.org/stable/user_guide.html`
- Licencia BSD open-source.

Scikit-learn

Machine learning



Regresiones en Machine Learning

Sckit-learn

- Las técnicas tanto estadísticas como computacionales en que se basa el Machine Learning han puesto fácilmente al alcance de cualquiera el uso de una familia de regresiones avanzadas. Las más usadas:
  - **lm.LinearRegression()**: admite como parámetro de la función **.predict()** el grado de la ecuación parabólica a ajustar.
  - **lm.Ridge()**: Ridge Regression o regularización de Tikhonov. Este estimador tiene soporte incorporado para regresión multivariante.
  - **lm.Lasso()**: Modelo lineal de Lasso con ajuste iterativo con regularización.
  - **lm.SGDRegressor()**: Stochastic Gradient Descent.
- Todas las funciones de sklearn de regresiones acabadas en CV, es que admiten *Cross-Validation*.
- En la documentación de scikit-learn se pueden consultar todas las posibilidades: `https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model`

Regresiones múltiples

sklearn.linear_model

sklearn.linear_model.LinearRegression() calcula un modelo de regresión lineal basado en la estimación por mínimos cuadráticos (OLS). Scikit-learn, en cambio, infiere la formula de la regresión múltiple, que pueden ser consultados con **.predict()**.

- **lm.fit()**: Ajuste a un modelo lineal
- **lm.predict()**: Predice Y usando el modelo lineal con coeficientes estimados
- **lm.score()**: Devuelve el coeficiente de determinación (R^2): como de bien los resultados observados son replicados por el modelo, como la proporción de la variación total de los resultados explicada por el modelo.

Para obtener la formula de la regresión:

- **lm.coef_**: Parametros de la regresión
- **lm.intercept_**: Valor independiente

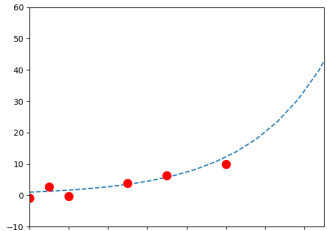
Sobreajuste: overfitting

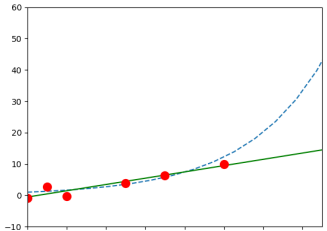
Veamos con otro ejemplo lo que significa el **overfitting** usando solo una variable explicativa para poder visualizarlo en una gráfica 2D.

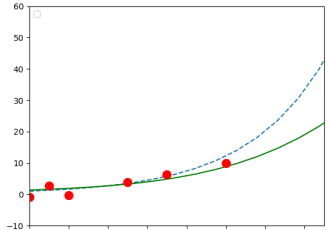
- Seleccionamos una serie de puntos (en rojo) creados con una función exponencial ($y = e^{2.5\dot{x}}$). Los puntos se calculan sobre esta curva (se muestra en todos las gráficas en línea azul discontinua) añadiéndose algo de ruido.
- Es decir, los puntos rojos equivaldrían a los puntos de entrenamiento del modelo, y la curva en azul discontinua sería la regla exacta que rige los datos (fórmula exponencial).
- Se estiman diferentes modelos bajo una regresión lineal (2), una regresión Ridge (3) y una regresión polinominal (4). Los residuos, aunque no se muestran, pueden apreciarse visualmente.
- El código de este ejemplo puede encontrarse en el fichero `Ejemplo_overfitting.ipynb`
- **Conclusión**: Mejores residuos no significa mejor modelo. Los modelos deben validarse con datos de entrenamiento y de test (validación cruzada).

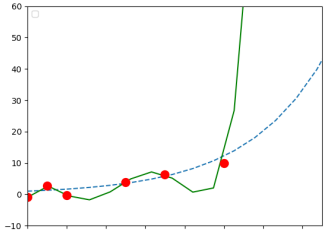
Sobreajuste: overfitting

(1) Datos en rojo y la fórmula exponencial para su creación en azul.
(2) Modelo de regresión lineal (3) Ridge regression (4) Polynomial regression









Cross-Validation

Validación cruzada

- La **validación cruzada** es el método estadístico utilizado en los modelos de *machine learning* para medir su capacidad predictiva.
  - Se divide los datos aleatoriamente en dos grupos: **datos de entrenamiento** y **datos de test**.
  - Ajusta un modelo en el conjunto de entrenamiento y lo evalúa en el conjunto de test.
  - Se obtiene el error de previsión para el modelo estimado (p.ej: MSE o Error cuadrático medio) y se compara con otros posibles modelos.
- Según el tipo de modelo y los datos disponibles, es practica habitual dividir 75%-25% nuestros datos (aunque algún autor trabaja con 50%-50%).
- Hay muchas formas de dividir el conjunto de datos inicial en dos partes (training, testing). Una posibilidad es extraer una muestra que forma el conjunto de training y colocarla en el conjunto de pruebas (testing). A esto se le llama validación cruzada (Leave-One-Out o Dejar uno fuera).
- Con N muestras, obtenemos N sets de training y pruebas.

Cross-Validation

Crear datos de entrenamiento

- En **Scikit-Learn** se dispone de la función **.model_selection.train_test_split(test_size)** *(por defecto test_size=0.25)*
- Ejemplo de división con scikit-learn:

```
1 from sklearn.model_selection import train_test_split
2 data = pd.read_csv('datos.csv')
3 train, test = train_test_split(data)
```

Cross-Validation

Ejemplo

```
1  from sklearn.model_selection import train_test_split
2  from sklearn import datasets, linear_model
3  data = datasets.load_boston()
4  X_train, X_test, y_train, y_test = train_test_split(data.data, data.
       target)
5
6  lm = linear_model.LinearRegression()
7  model = lm.fit(X_train, y_train)
8  predictions = lm.predict(X_test)
9
10 print(lm.score(X_train, y_train))
11 print(lm.score(X_test, y_test))
12 #print(lm.coef_, lm.intercept_)
```

```
0.7513466212833374
0.7040640680312542
```

Ejercicio 5 (cont)

Ejercicio 5. Regresión con machine learning - Precio vivienda en Boston

Con los mismos datos del ejercicio anterior, el set de datos de Boston (BostonHousing.csv), se pide estimar el precio de la vivienda aplicando el enfoque propio de machine learning:

- Dividir datos en datos en entrenamiento y test.
- Entrenar el modelo utilizando las siguientes funciones de regresión [a]:
  - LinearRegression()
  - RidgeCV()
  - SGDRegressor()
  - LassoCV()
- Validar el ajuste con los datos de test.

(Es decir, se pide replicar el ejemplo anterior modificando la función de regresión, y comparar los resultados).

[a] `https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model`

Scikit-learn: árbol de decisión

https://scikit-learn.org/stable/tutorial/machine_learning_map/