Applied Machine Learning for Business and Economics

AMLBE

Profesor : Raúl Muñoz R.

Entrenamiento de Modelos

- En esta etapa comenzaremos revisitando uno de los modelos más simples el modelo de regresión lineal, sin embargo revisaremos 2 formas muy diferentes de calibración de sus parámetros:

  - Utilizando directamente la formula cerrada que directamente calcula los parámetros del modelo que tienen el mejor fit del modelo en el set de entrenamiento i.e. los parámetros del modelo que minimizan la función de costos sobre el set de entrenamiento
  - Utilizando un approach iterativo llamado Gradiente Descendente (GD), que gradualmente actualiza los parámetros del modelo de manera tal que minimiza la función de costos sobre el set de entrenamiento, eventualmente convergiendo al mismo set de parámetros del primer método

- **Modelo de Regresión Lineal** $\hat{y} = \boldsymbol{\theta}_0 + \boldsymbol{\theta}_1 x_1 + \boldsymbol{\theta}_2 x_2 + \cdots + \boldsymbol{\theta}_n x_n$
- Donde $\hat{y}$ es el valor predicho
- n es el número de atributos o características
- x_i es el valor del i-ésimo atributo
- $\boldsymbol{\theta}_j$ es el j-ésimo parámetro del modelo

Entrenamiento de Modelos: La Ecuación Normal

- Lo anterior puede ser escrito en su notación matricial de la siguiente manera: $\hat{y} = h_{\boldsymbol{\theta}}(X) = \Theta^T \cdot X$
- Donde Θ corresponde al vector de parámetros
- Θ^T es la matriz transpuesta de Θ
- X es el vector que contiene el valor de los atributos del modelo x_0 siempre es igual a 1
- $\Theta^T \cdot X$ es el producto punto entre Θ^T y X
- Para la cual el entrenamiento habitual se utiliza la siguiente función de costo:

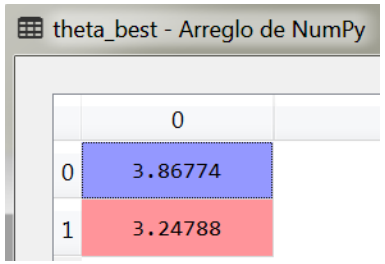
$$MSE(X, h_{\boldsymbol{\theta}}) = \frac{1}{m}\sum_{i=1}^{m}(\Theta^T \cdot x^{(i)} - y^{(i)})^2$$

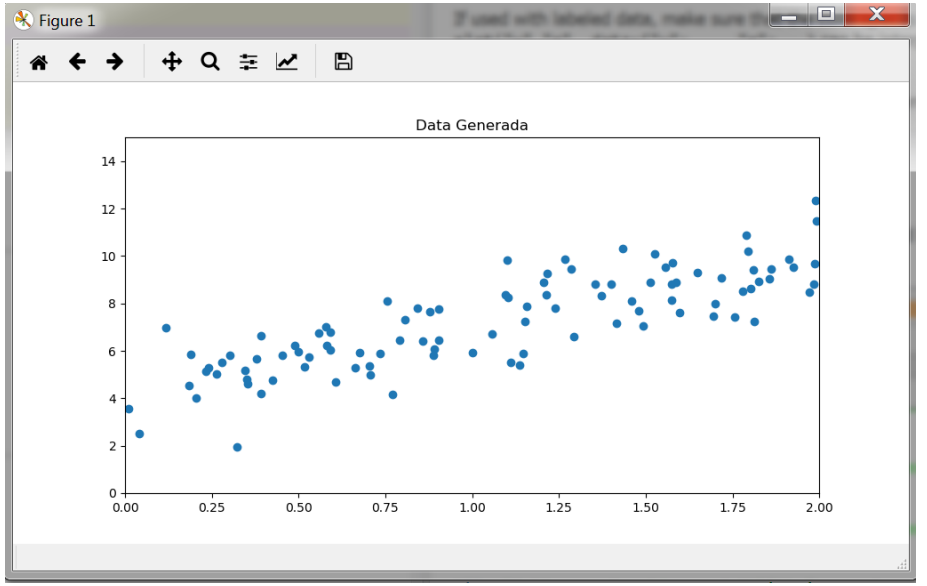
- La cual tiene solución cerrada denominada «ecuación normal» $\widehat{\boldsymbol{\theta}} = (X^T X)^{-1} X^T y$
- Donde *y* es el vector de variables target

Entrenamiento de Modelos: La Ecuación Normal

- Generemos algunos datos lineales para recuperar sus parámetros estructurales:

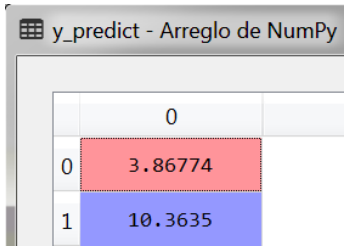
```
X=2*np.random.rand(100,1)
y=4+3*X+np.random.randn(100,1)
X_b=np.c_[np.ones((100,1)),X]
theta_best=np.dot(np.dot(np.linalg.inv(np.dot(X_b.T,X_b)),X_b.T),y)
```





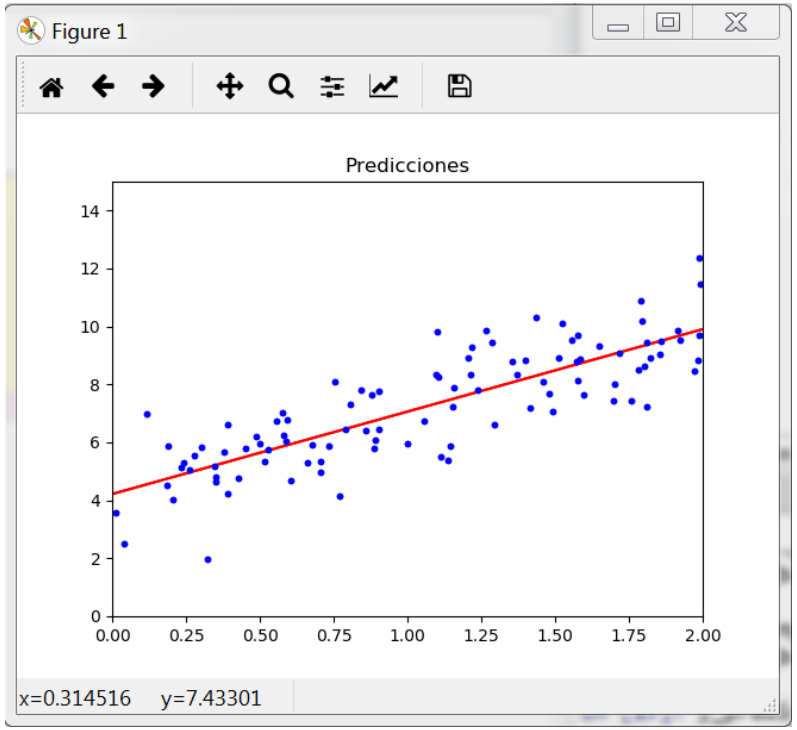
- Hubiéramos esperado obtener $\boldsymbol{\theta}_0 = 4$ y $\boldsymbol{\theta}_1 = 3$ en vez de $\boldsymbol{\theta}_0 = 3.86$ y $\boldsymbol{\theta}_1 = 3.24$, pero el «ruido» hizo imposible recuperar los parámetros exactos
- Ahora podemos generar predicciones con el vector $\hat{\boldsymbol{\theta}}$

```
X_new=np.array([[0],[2]])
X_new_b=np.c_[np.ones((2,1)),X_new]
y_predict=np.dot(X_new_b,theta_best)
```



Entrenamiento de Modelos: La Ecuación Normal

- Generemos algunos datos lineales para recuperar sus parámetros estructurales:



Entrenamiento de Modelos: Métricas de Bondad de Ajuste

- Métricas de bondad de ajuste:

- R2: Corresponde al porcentaje de varianza explicada por el modelo:

$$R^2 = \frac{(\Theta^T X^T)y - n\hat{y}\_prom^2}{y^T y - n\hat{y}\_prom^2}$$

- R2 ajustado: Mismo concepto anterior, aunque penaliza el overfitting

$$R^2\_Theil = 1 - \frac{(1 - R^2)(n-1)}{(n-k)}$$

$$R^2\_Goldberger = \left(1 - \frac{k}{n}\right) R^2$$

Entrenamiento de Modelos: Métricas de Bondad de Ajuste

- Métricas de bondad de ajuste:

- T-test: Corresponde al test estadístico de significancia individual, corresponde al test de hipótesis de que el parámetro estimado tiene el valor cero dentro de su intervalo de confianza, este test sigue una distribución T de Student con n-k grados de libertad

$$v = var(y - \hat{y})$$

$$sig_{\boldsymbol{\theta}} = \sqrt{diag((X^T X)^{-1}\, v)}$$

$$\text{T-test} = \frac{\Theta}{sig_{\boldsymbol{\theta}}}$$

- F-Test: Corresponde al test de significancia conjunta, es decir testea que todos los parámetros sean simultáneamente iguales a cero, puede ser utilizado también para evaluar la contribución incremental de una variable explicativa. Ésta sigue una distribución F de Fischer con n-1 y n-k grados de libertad

$$F - Test = \frac{\frac{R^2}{k-1}}{\frac{R^2}{n-k}}$$

Entrenamiento de Modelos: Métricas de Bondad de Ajuste

- Métricas de bondad de ajuste:

```
import scipy.stats as s
import numpy as np

n=float(len(X_b))
k=float(int(X_b.shape[1]))
theta_best=np.dot(np.dot(np.linalg.inv(np.dot(X_b.T,X_b)),X_b.T),y)
y_est=np.dot(X_b,theta_best)
y_prom=y_est.mean()
r2=(np.dot(np.dot(theta_best.T,X_b.T),y)-n*y_prom**2)/(np.dot(y.T,y)-n*y_prom**2)
r2atheil=1-(1-r2)*(n-1)/(n-k)
r2agoldberger=(1-(k/n))*r2
f_test=(r2/(k-1))/((1-r2)/(n - k))
v=np.var(y-y_est)
v_theta=np.linalg.inv(np.dot(X_b.T,X_b))*v
sig_theta=np.sqrt(np.diag(v_theta)).reshape(len(v_theta),1)
t_test=theta_best/sig_theta
te=np.sqrt(v)

ndc=0.05
tc=s.t.ppf(1-ndc,n-k)
fc=s.f.ppf(1-ndc,n-1,n-k)
invtc=s.t.cdf(tc,n-k)
invfc=s.f.cdf(fc,n-1,n-k)
```

theta_best	float64	(2L, 1L)	array([[3.86774363], [3.2478776]])
r2	float64	(1L, 1L)	array([[0.80018664]])
r2agoldberger	float64	(1L, 1L)	array([[0.78418291]])
r2atheil	float64	(1L, 1L)	array([[0.79814773]])
sig_theta	float64	(2L, 1L)	array([[0.1770017], [0.16229916]])
t_test	float64	(2L, 1L)	array([[21.85144944], [20.01167261]])
f_test	float64	(1L, 1L)	array([[392.4576999]])
tc	float64	1	1.6605512170440568
fc	float64	1	1.3954807011947474
invfc	float64	1	0.95000000000000029
invtc	float64	1	0.94999999999781592

Entrenamiento de Modelos Stats

- El equivalente Stats:

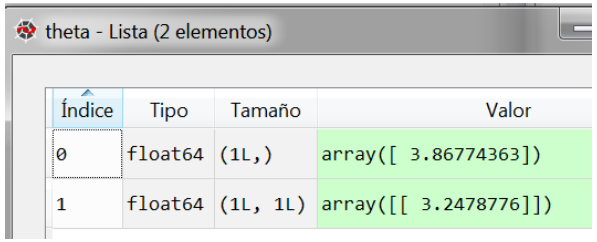
```
import statsmodels.formula.api as smf

est = smf.OLS(y,X_b).fit()
est.summary()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.800
Model:                            OLS   Adj. R-squared:                  0.798
Method:                 Least Squares   F-statistic:                     392.5
Date:                Sat, 14 Oct 2017   Prob (F-statistic):           4.82e-36
Time:                        16:17:59   Log-Likelihood:                -127.98
No. Observations:                 100   AIC:                             260.0
Df Residuals:                      98   BIC:                             265.2
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          3.8677      0.179     21.632      0.000       3.513       4.223
x1             3.2479      0.164     19.811      0.000       2.923       3.573
==============================================================================
Omnibus:                        5.978   Durbin-Watson:                   2.106
Prob(Omnibus):                  0.050   Jarque-Bera (JB):                5.719
Skew:                          -0.424   Prob(JB):                       0.0573
Kurtosis:                       3.808   Cond. No.                         3.82
==============================================================================
```

- El equivalente Scikit-Learn:

```
theta=[]
from sklearn.linear_model import LinearRegression
lin_reg=LinearRegression()
lin_reg=lin_reg.fit(X,y)
theta.append(lin_reg.intercept_)
theta.append(lin_reg.coef_)
lin_reg.predict(X_new)
```




```
array([[  3.86774363],
       [ 10.36349883]])
```

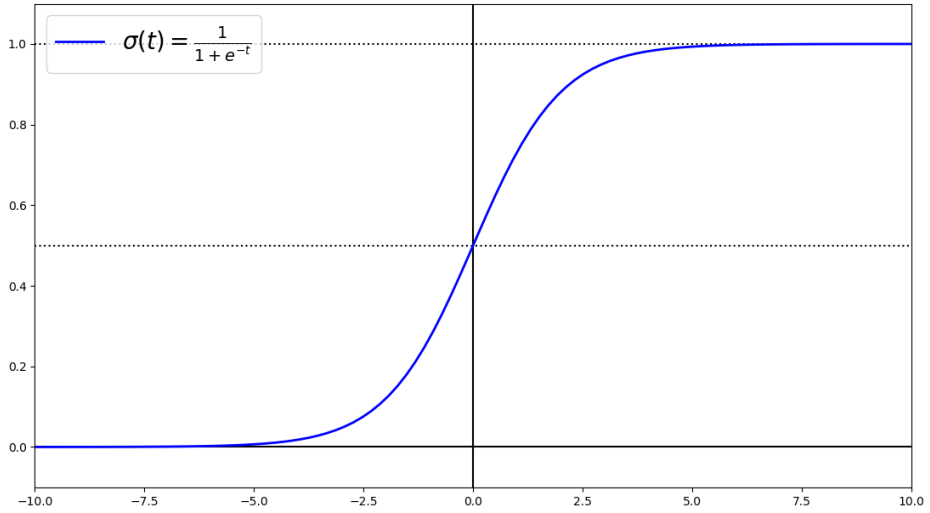
- Complejidad computacional, la ecuación normal requiere el calculo de $(X^T X)^{-1}$. La complejidad computacional de invertir tal matriz es típicamente entre O($n^{2.4}$) y O(n^3). En otras palabras si duplicamos el numero de atributos de un modelo el tiempo de cálculo se multiplicaría entre $2^{2.4} = 5.3\ y\ 2^3 = 8$
- La ecuación normal puede llegar a volverse muy lenta cuando el número de atributos es grande (e.g. 100.000)
- Por el lado positivo esta ecuación es lineal respecto del número de instancias en el training set (esto es *O(m)*), de manera tal que puede manejar grandes training sets de manera eficiente
- Adicionalmente una vez que ya se ha entrenado el modelo de regresión con la ecuación normal o algún otro algoritmo las predicciones son muy rápidas, la complejidad computacional es lineal respecto de ambas: el número de instancias a las cuales se requiere hacer predicciones y el numero de atributos

- Algunos algoritmos de regresión pueden ser utilizados también para clasificación
- La regresión logística o logit es utilizada comúnmente para estimar la probabilidad de que una instancia pertenezca a una clase particular
- Si la probabilidad es mayor que 50% el modelo predice que la instancia pertenece a la clase «1» de lo contrario a la clase «0», lo que lo hace un clasificador binario
- En este modelo, en vez de generar el output directamente, como es el caso de la regresión lineal, en este caso el output es la función logística de dicho resultado

$$\hat{p} = h_0(X) = \sigma(\boldsymbol{\theta}^T X)$$

- La función logística $\sigma()$ es una función sigmoidea (con forma de «S») que genera un output entre 0 y 1

$$\sigma(t) = \frac{1}{1+\exp(-t)}$$



$$\hat{y} = \begin{cases} 0 \; si \; \hat{p} < 0.5 \\ 1 \; si \; \hat{p} \geq 0.5 \end{cases}$$

- Notar que $\sigma(t)$<0.5 cuando t<0 y $\sigma(t)$≥0.5 cuando t≥0, de tal modo que la regresión logística predice 1 si $\boldsymbol{\theta}^T X$ es positivo y 0 si es negativo

- **Entrenamiento y Función de Costo:**

- Se requiere establecer el vector de parámetros $\boldsymbol{\theta}$, tal que el modelo estime una alta probabilidad para las instancias positivas (y=1) y bajas probabilidades para las instancias negativas (y=0)
- Esta idea es capturada en la función de costo:

$$C(\boldsymbol{\theta}) = \begin{cases} -\log(\hat{p}) \; si \; y = 1 \\ -\log(1-\hat{p}) \; si \; y = 0 \end{cases}$$

- Lo anterior tiene sentido dado que –log(t) crece mucho cuando t se aproxima a 0, de modo que la función de costo, será alta si el modelo estima una probabilidad cercana a 0 para una instancia positiva
- Asimismo el costo será alto si el modelo estima una probabilidad cercana a 1 para una instancia negativa
- Por otro lado –log(t) es cercana a 0 cuando t es cercano a 1, de manera que el costo será cercano a 0 cuando la probabilidad estimada es cercana a 0 para una instancia negativa o cercano a 1 para una instancia positiva, que es precisamente el objetivo

- **Entrenamiento y Función de Costo:**

- La función de costo sobre todo el set de entrenamiento es simplemente el costo promedio sobre todas las instancias de entrenamiento

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right)\log(1 - \hat{p}^{(i)})]$$

- La mala noticia es que no hay una ecuación de forma cerrada que calcule el valor de $\boldsymbol{\theta}$ que minimice la función de costo (equivalente a la ecuación normal)
- Pero la buena noticia es que la función es convexa de modo tal que el gradiente descendente (u otro algoritmo de optimización) encontrarán el mínimo global la derivada parcial de la función de costo respecto del j-ésimo parámetro viene dado por:

$$\frac{\partial}{\partial \boldsymbol{\theta}_j} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}(\sigma\left(\boldsymbol{\theta}^T X^{(i)}\right) - y^{(i)})\, x_j^{(i)}$$

- Esta ecuación calcula el error de predicción para cada instancia y lo multiplica por el valor de la j-ésima valor del atributo y luego calcula el promedio sobre todas las instancias de entrenamiento

Entrenamiento de Modelos: Regresión Logística sklearn

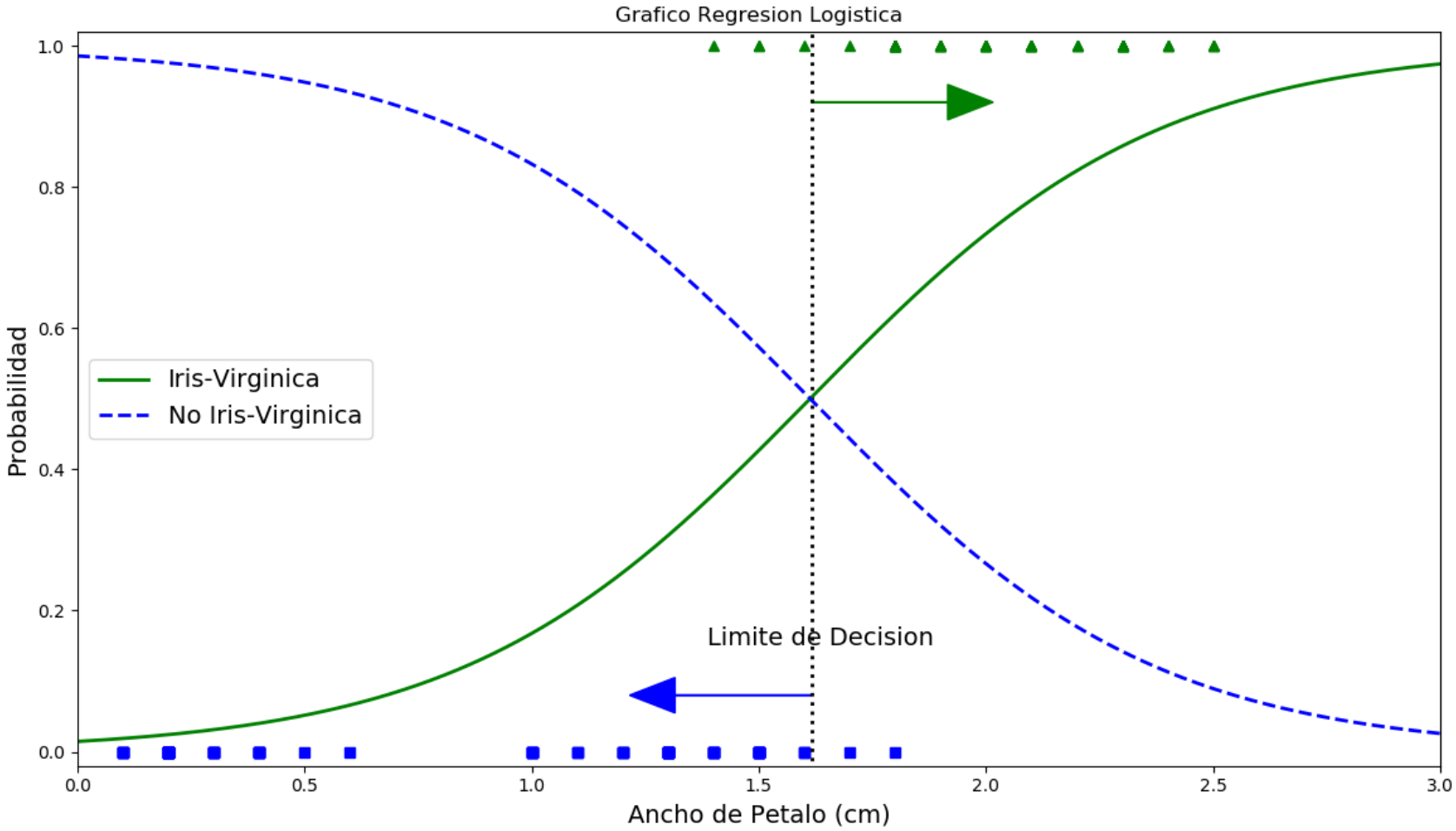
```
from sklearn import datasets
iris = datasets.load_iris()
list(iris.keys())
print(iris.DESCR)

X = iris["data"][:, 3:]  # ancho de pétalo
y = (iris["target"] == 2).astype(np.int)  # 1 si Iris-Virginica, si no 0

from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(X, y)

X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
decision_boundary = X_new[y_proba[:, 1] >= 0.5][0]
```

Entrenamiento de Modelos: Regresión Logística sklearn



Entrenamiento de Modelos: Métricas de Bondad de Ajuste en Clasificadores

- Evaluar un clasificador es a menudo más engañoso que evaluar una regresión, por lo que vamos a revisar variadas métricas para esta tarea

- **Cross-Validation:**

- El proceso de validación cruzada consiste en hacer particiones de la data separando, por medio de algún criterio, en muestras de prueba y entrenamiento
- Uno de ellos (dentro de muchos) es el denominado K-folds, donde se divide toda la muestra en k muestras aleatorias de igual tamaño llamadas «folds»
- Para cada una de las k iteraciones una de las muestras es retenida como test set mientras que las k-1 remanentes conforman todas un train test
- Luego un modelo es entrenado en cada una de las k-folds y su performance es evaluado en su específico k-ésimo test

- **Accuracy Score:**

- Corresponde simplemente a la precisión media calculada como la razón entre las clasificaciones correctamente realizadas y el total de clasificaciones realizadas

- Esta métrica por si sola puede ser engañosa en los casos donde las muestras están desbalanceadas e.g. supongamos que de 100 casos sólo 5 no corresponden a la clasificación 1, en este caso un simple un «dumb» classifier que clasifique siempre la clase 1 , obtendría un score de 95%
- Dado esto, en problemas de clasificación, debemos complementar con otras métricas

$$Accuracy\ Score = \frac{TP + TN}{TP + FP + TN + FN}$$

- **Confusion Matrix:**

- Una mucho más completa forma de evaluar el desempeño del clasificador es por medio de la matriz de confusión
- La idea general es:

	Clase Predicha	
Clase Real	**True Positives (TP)**	**False Negatives (FN)**
	False Positives (FP)	True Negatives (TN)

- **Precisión:**

- Esta métrica calcula la precisión sólo de las predicciones de instancias positivas

$$Precision = \frac{TP}{TP + FP}$$

- **Recall (Sensitivity-True Positive Rate TPR):**
- A modo complementario a la medición anterior

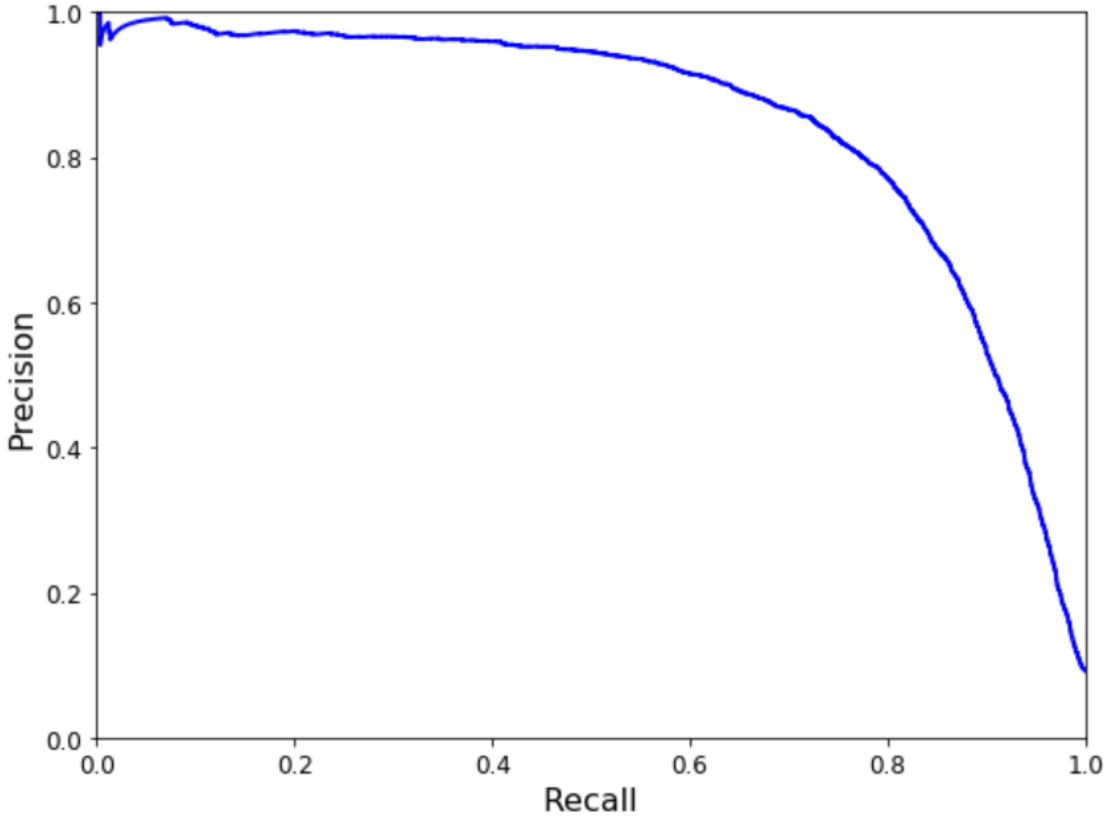
$$TPR = \frac{TP}{TP+FN} = Recall$$

- **F1 Score:**
- Si buscamos una simple manera de comparar clasificadores es conveniente contar con una métrica que combine precisión y recall, el cual corresponde a la media armónica de Precisión y Recall
- La media armónica a diferencia de la media regular, que da igual peso a todos los valores, la media armónica da mucho más peso a los valores bajos
- Como resultado el clasificador obtendrá un alto F1 score si ambos, Precisión y Recall son altos

$$F1 = \frac{2}{\frac{1}{Precisión} + \frac{1}{Recall}} = 2\frac{Precisión\ x\ Recall}{Precisión + Recall} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

Entrenamiento de Modelos: Métricas de Bondad de Ajuste en Clasificadores

- Trade-Off Precision/Recall



- El F1 Score favorece aquellos clasificadores que tienen similar Precisión y Recall, lo cual no es siempre un atributo deseado, por ejemplo si estamos entrenando un clasificador para detectar os videos que son seguros para los niños probablemente preferiremos un clasificador que rechace muchos buenos videos (Recall Bajo) pero que acepte sólo los seguros (Precisión Alta)
- En vez de un clasificador que tenga alto Recall pero acepte videos inseguros (clase 1 es «Video Seguro» y clase 0 es «Video Noseguro»)

	Clase Predicha	
Clase Real	**Video Seguro/Clasificado como Seguro (TP)**	**Video Seguro/Clasificado como No Seguro (FN)**
	Video no Seguro/Clasificado como Seguro (FP)	Video No Seguro/Clasificado como No Seguro (TN)

$$Precision = \frac{TP}{TP + FP}$$

- En el ejemplo de arriba preferiremos Precisión, dada la asimetría en el costo de equivocarse Relevancia(FP)>Relevancia(FN) ergo Relevancia(Precisión)>Relevancia(Recall)

- Que indicador preferiría en este caso ? (clase 1 es «Ladrón de Tiendas» y clase 0 es «Comprador Normal»)

Clase Predicha

Clase Real	**Ladrón de Tiendas /Clasificado como Ladrón de Tiendas (TP)**	**Ladrón de Tiendas /Clasificado como Comprador Normal (FN)**
	Comprador Normal/Clasificado como Ladrón de Tiendas (FP)	Comprador Normal/Clasificado como Comprador Normal (TN)

$$TPR = \frac{TP}{TP+FN} = Recall$$

- En el ejemplo de arriba preferiremos Recall, dada la asimetría en el costo de equivocarse Relevancia(FN)>Relevancia(FP) ergo Relevancia(Recall)>Relevancia(Precisión)
- O de otro modo, los vigilantes recibirán algunas falsas alarmas pero casi todos los Ladrones serán atrapados

- **La Curva ROC (Receiver Operating Characteristic):**

- Corresponde al gráfico del Recall o True Positive Rate (TPR) contra el False Positive Rate (FPR) donde:

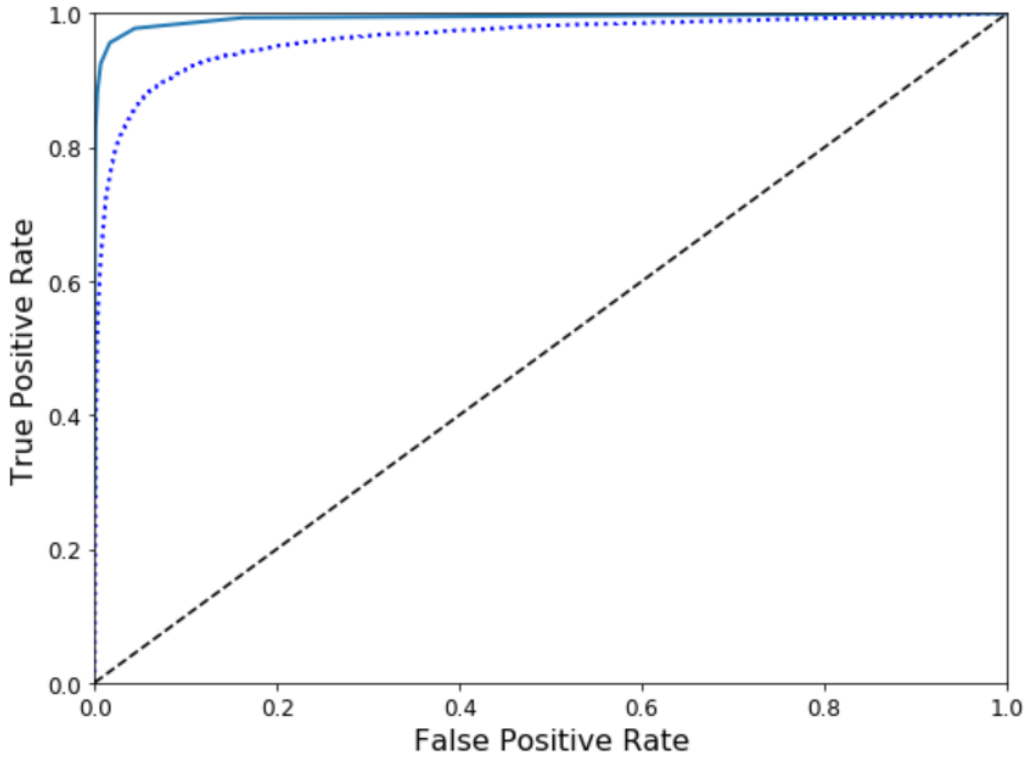
$$TPR = \frac{TP}{TP + FN} = Recall$$

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} = 1 - Specificity$$

$$ROC = \frac{TPR}{FPR}$$

- En este caso, al igual que en la relación Precision/Recall, i.e. mientras mayor es el recall (TPR), mayor es la cantidad de falsos positivos (FPR) generados por el clasificador
- Mientras más nos alejemos de la línea de 45° (curva ROC que generaría un clasificador puramente aleatorio), mejor será el clasificador

- **La Curva ROC (Receiver Operating Characteristic):**



- **ROC AUC Score:**

- Otra forma de comparar clasificadores es por medio el cálculo del área bajo la curva (Area Under the Curve, AUC), un clasificador perfecto, tendrá un ROC AUC score igual a 1, mientras que un clasificador puramente aleatorio tendrá un ROC AUC Score de 0.5

- **Implementación Sklearn:**

- Se requiere importar los siguientes componentes:

```
from sklearn import cross_validation
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
```

→ Métodos de Validación Cruzada
→ Métricas para evaluación
→ Modelos Lineales

- Ahora vamos al Lab …

Entrenamiento de Modelos: Clasificación Multinomial

- **Clasificación Multinomial:**

- Hasta el minuto hemos analizado el modelo de regresión logística como un método de clasificación binaria donde sólo 2 clases son posibles
- Algunos algoritmos (como el Random Forest o Naive Bayes classifiers) son capaces de manejar múltiples clases directamente
- Otros (como Support Vector Machines o Clasificadores Lineales) son estrictamente clasificadores binarios
- Sin embargo existen maneras en que puede realizarse clasificación multinomial por medio de múltiples clasificadiones binarias

- **One versus Rest (OvR):**

- Consiste en la creación de un clasificador separando las n clases entre la i-ésima clase asignando valor 1 y 0 para las n-1 restantes
- Así sucesivamente se entrena una clasificador para las n clases
- Finalmente cuando se requiere de clasificar una nueva instancia, se selecciona aquella clase que tenga el mayor score de clasificación

Entrenamiento de Modelos: Clasificación Multinomial

- **One versus One (Ovo):**

- Consiste en consiste en crear un clasificador binario para cada par de clases y sus respectivas combinaciones
- Si tenemos n clases existirán n*(n-1)/2 clasificadores e.g. si tuvieramos 10 clases existirían 45 clasificadores
- Cuando se quiere clasificar una nueva instancia se clasificará en aquella categoría que gane una mayor cantidad de duelos

- **Cual utilizar ?**

- Una ventaja de Ovo versus OvR es que cada clasificador debe ser sólo entrenado sobre la parte del set de entrenamiento entre la cual se quiere generar la distinción
- Por lo anterior si bien se requieren más clasificaciones que en OvR, se realizará sobre una pequeña fracción del data set
- Algunos algoritmos (como SVM) escalan pobremente con el tamaño del set de entrenamiento de modo que para ese tipo de algoritmos se preferirá Ovo dado que es más eficiente entrenar múltiples clasificadores sobre pequeños sets de entrenamiento
- Sin embargo para la mayoría de los clasificadores binarios OvR generará un mejor desempeño

Entrenamiento de Modelos: Clasificación Multinomial

- Sklearn detecta automáticamente cuando un clasificador binario intenta ser utilizado para una tarea multinomial y por default corre OvR, con excepción de los Support Vector Machines, donde por default utiliza OvO

class sklearn.linear_model. **LogisticRegression** (*penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0, warm_start=False, n_jobs=1*) ¶ [source]

```
from sklearn import metrics
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression

method = OneVsRestClassifier(LogisticRegression())
model=method.fit(X,y)
y_pred=model.predict(X[len(X)-1,:].reshape(1,int(np.shape(X[len(X)-1,:])[0])))
y_proba=model.predict_proba(X[len(X)-1,:].reshape(1,int(np.shape(X[len(X)-1,:])[0])))
y_real=y[len(y)-1]
```

y_pred	float64	(1L,)	array([5.])
y_proba	float64	(1L, 9L)	array([[0.04824477,
y_real	float64	1	4.0

y_proba - Arreglo de NumPy

	0	1	2	3	4	5	6	7	8
0	0.0482448	0.0390809	0.0818087	0.13998	0.188014	0.199889	0.129563	0.131524	0.0418966