Machine learning y Deep learning con Python: Lectura 3

Ing. Pedro Rotta

Universidad de Piura - Vida Universitaria

Enero-2022

Aprendizaje Supervizado

El aprendizaje supervisado es aquel que necesita etiquetar o definir un objetivo respecto a los datos de entrada en un conjunto de datos para entrenamiento.

Aprendizaje Supervizado

El aprendizaje supervisado es aquel que necesita etiquetar o definir un objetivo respecto a los datos de entrada en un conjunto de datos para entrenamiento.

Existen dos problemas centrales que resuelve el aprendizaje supervizado y son: la clasificación y la regresión.

Aprendizaje Supervizado

El aprendizaje supervisado es aquel que necesita etiquetar o definir un objetivo respecto a los datos de entrada en un conjunto de datos para entrenamiento.

Existen dos problemas centrales que resuelve el aprendizaje supervizado y son: la clasificación y la regresión.

En la **clasificación** el objetivo es predecir una **etiqueta** para cada entrada de datos. La clasificación se puede separar en clasificación binaria y clasificación multiclase.

Aprendizaje Supervizado

El aprendizaje supervisado es aquel que necesita etiquetar o definir un objetivo respecto a los datos de entrada en un conjunto de datos para entrenamiento.

Existen dos problemas centrales que resuelve el aprendizaje supervizado y son: la clasificación y la regresión.

En la **clasificación** el objetivo es predecir una **etiqueta** para cada entrada de datos. La clasificación se puede separar en clasificación binaria y clasificación multiclase.

En la **regresión** el objetivo es predecir un valor determinado para un conjunto de datos. La regresión puede ser lineal y polinómica.

Generalización, Sobreajuste y Subajuste

Cuando generamos un modelo de Machine learning con un conjunto de datos de entrenamiento, que predice datos que no ha visto, es decir, predice bien, datos nuevos, decimos que este modelo permite **generalizar** la data de entrenamiento a la de prueba. El objetivo es obtener un modelo que generalice con un alto grado de exactitud.

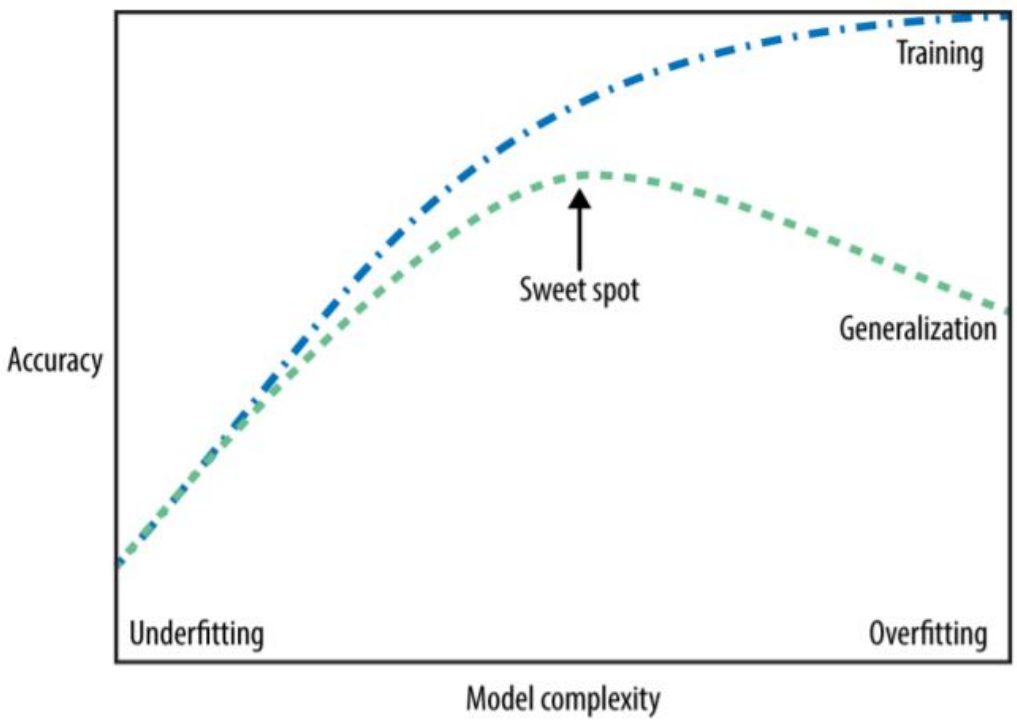
El primer caso de falla, sucede cuando hacemos un modelo complejo, que prediga el 100% del conjunto de entrenamiento. Este tipo de "super modelo" no trabaja bien con nuevos datos. En este caso, nos encontramos en un caso de sobreajuste.

El segundo caso de falla, sucede cuando hacemos un modelo muy simple, que no funciona bien para el conjunto de entrenamiento, es decir tenemos una precisión baja (Entre 5% y 45%). A este tipo de modelos se le conoce como modelos subajustados.

Generalización, Sobreajuste y Subajuste

Hay que tener en cuenta un trade-off entre complejidad del modelo y generalización para poder lograr un modelo correcto, es decir el modelo óptimo de predicción inteligencia artificial.

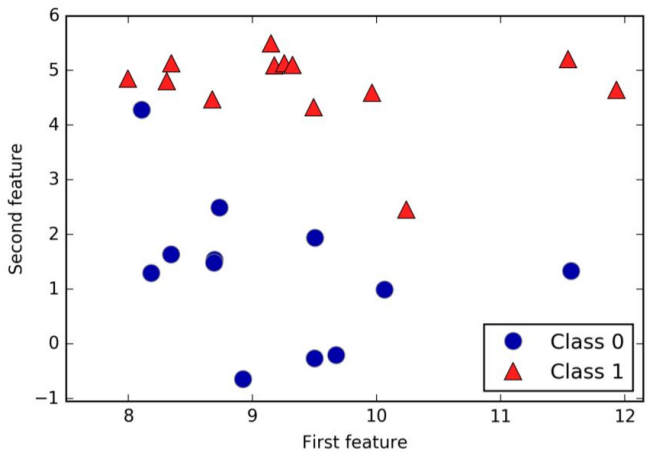
Por ejemplo, en la siguiente figura, se expresa la relación que existe entre complejidad y sobreajuste.

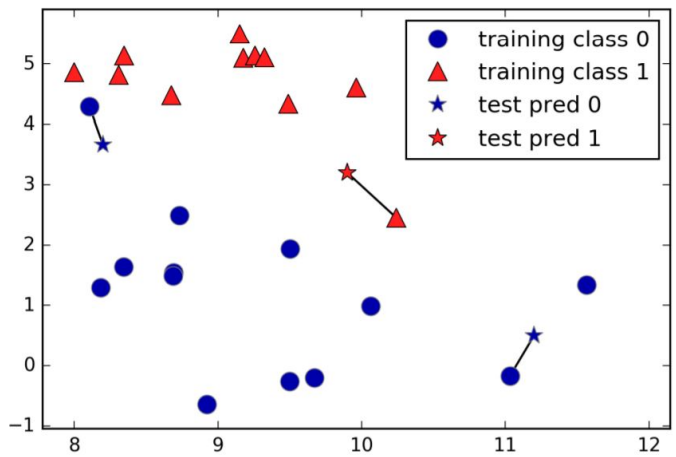


Los K-Vecinos más cercanos (KNN)

Los K-Vecinos más cercanos aka KNN (k-Nearest Neighbors) es el algoritmo de clasificación más simple de definir.

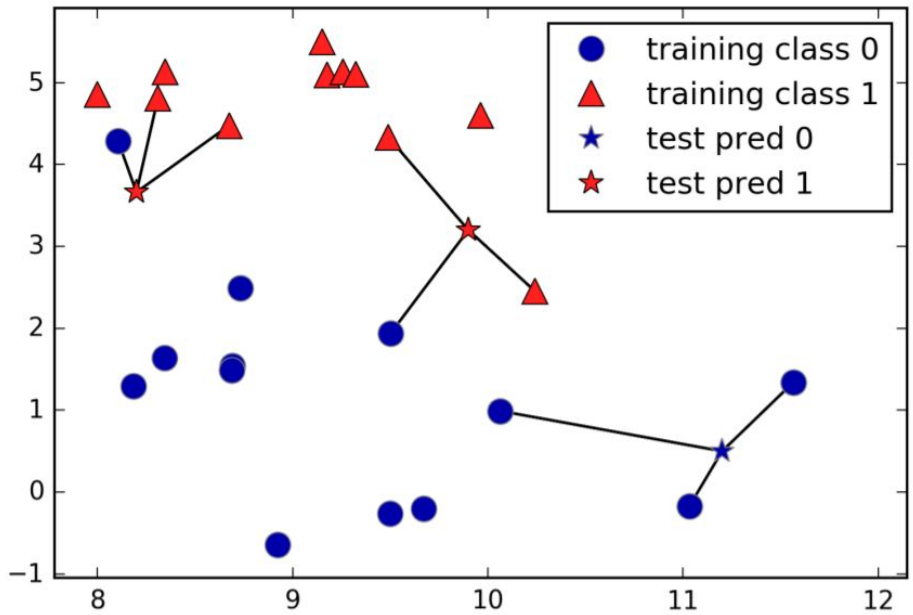
En su versión más sencilla de definir, el algoritmo K-NN considera exactamente un vecino cercano, que es el vecino de la data de entrenamiento más cercano. Por lo tanto la predicción será que la salida es dada por este dato de entrenamiento.





Los K-Vecinos más cercanos (KNN)

En el caso k= 3, el resultado estará determinado por los datos de entrenamiento más cercanos que más se repitan como se ve en la figura



Donde este algoritmo, se puede extrapolar a n clases, aunque acá se muestra para clasificación binaria. El objetivo es que si hay varias clases, la clase que más se repita en el entorno cercano, será la considerada.

KNN: Hiperparámetros y métricas

El hiperparámetro a considerar en este tipo de algoritmo es **K** que representa la cantidad de vecinos que se van a analizar.

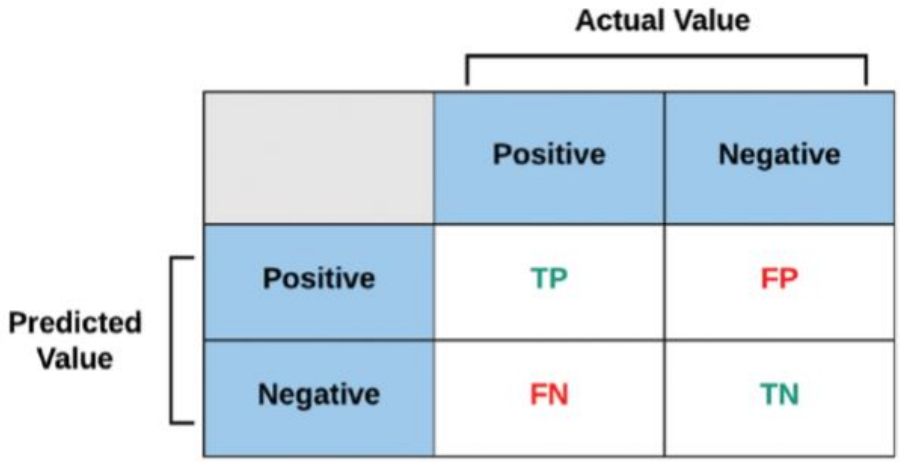
Para evaluar el modelo, optamos por las métricas de **total score** que es el porcentaje de datos de prueba correctamente definidos y se calcula como:

$$score(\%) = \frac{valores\_correctos}{datos\_totales} x100$$

También se puede optar por **la matriz de confusión**, el valor de **precisión** y el **recall**

KNN: Hiperparámetros y métricas

La matriz de confusión mide la relación entre los valores reales y los valores predichos clasificándolos entre los **verdaderos positivos** y los **verdaderos negativos**



Donde los **verdaderos positivos** son el número de muestras predichas como True, cuando su valor es True y los **falsos positivos** son los que han sido predecidos como True cuando en realidad son False. Así recíprocamente para los **falsos negativos** y los **verdaderos negativos**. Para cualquiera 2 clases True y False.

KNN:Hiperparámetros y métricas

Donde para estas 4 clases de datos, las métricas quedan definidas por:

$$score(\%) = \frac{TP + TN}{TP + TN + FP + FN} x100$$

$$precision(\%) = \frac{TP}{TP + FP} x100$$

$$recall(\%) = \frac{TP}{TP + FN} x100$$

Para entender más a fondo, veremos un ejemplo práctico

Métricas para clasificación Binaria

Supongamos un caso de análisis donde medimos si alguien tiene un caso de enfermedad maligna o benigna , por lo que tenemos dos clases:

Clase 1: ??
Clase 2: ??

Métricas para clasificación Binaria

Supongamos un caso de análisis donde medimos si alguien tiene un caso de enfermedad maligna o benigna , por lo que tenemos dos clases:

Clase 1: ??
Clase 2: ??

Luego de realizar el entrenamiento y validación de datos se tienen los siguientes resultados:

Datos del conjunto de entrenamiento:

Clase 1: 20 valores
Clase 2: 30 valores

Métricas para clasificación Binaria

Supongamos un caso de análisis donde medimos si alguien tiene un caso de enfermedad maligna o benigna , por lo que tenemos dos clases:

Clase 1: ??
Clase 2: ??

Luego de realizar el entrenamiento y validación de datos se tienen los siguientes resultados:

Datos del conjunto de entrenamiento:

Clase 1: 20 valores
Clase 2: 30 valores

Datos de predicción:

Clase 1 que son clase 1 : 15

Métricas para clasificación Binaria

Supongamos un caso de análisis donde medimos si alguien tiene un caso de enfermedad maligna o benigna , por lo que tenemos dos clases:

Clase 1: ??
Clase 2: ??

Luego de realizar el entrenamiento y validación de datos se tienen los siguientes resultados:

Datos del conjunto de entrenamiento:

Clase 1: 20 valores
Clase 2: 30 valores

Datos de predicción:

Clase 1 que son clase 1 : 15(Verdaderos positivos)
Clase 1 que son clase 2 : 5

Métricas para clasificación Binaria

Supongamos un caso de análisis donde medimos si alguien tiene un caso de enfermedad maligna o benigna , por lo que tenemos dos clases:

Clase 1: ??
Clase 2: ??

Luego de realizar el entrenamiento y validación de datos se tienen los siguientes resultados:

Datos del conjunto de entrenamiento:

Clase 1: 20 valores
Clase 2: 30 valores

Datos de predicción:

Clase 1 que son clase 1 : 15(Verdaderos positivos)
Clase 1 que son clase 2 : 5(Falsos positivos)
Clase 2 que son clase 1 : 5

Métricas para clasificación Binaria

Supongamos un caso de análisis donde medimos si alguien tiene un caso de enfermedad maligna o benigna , por lo que tenemos dos clases:

Clase 1: ??
Clase 2: ??

Luego de realizar el entrenamiento y validación de datos se tienen los siguientes resultados:

Datos del conjunto de entrenamiento:

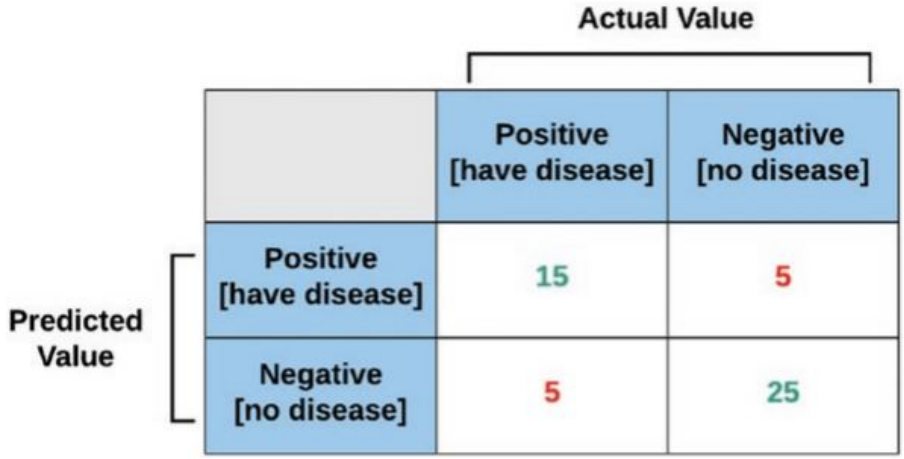
Clase 1: 20 valores
Clase 2: 30 valores

Datos de predicción:

Clase 1 que son clase 1 : 15(Verdaderos positivos)
Clase 1 que son clase 2 : 5(Falsos positivos)
Clase 2 que son clase 1 : 5(Falsos negativos)
Clase 2 que son clase 2 : 25 (Verdaderos negativos)

Métricas para clasificación Binaria

Entonces, la matriz de confusión se formará así :



$$score(\%) = \frac{15 + 25}{15 + 5 + 5 + 25} x100 = 80\%$$

$$precision(\%) = \frac{15}{15 + 5} x100 = 75\%$$

$$recall(\%) = \frac{15}{15 + 5} x100 = 75\%$$

Computarización del algoritmo KNN

Para computarizar el modelo de usaremos la librería scikit learn.

```
from sklearn.neighbors import KNeighborsClassifier
```

Creamos el modelo, con la siguiente sintaxis:

```
modelknn = KNeighborsClassifier(n_neighbors = 3)
```

Entrenamos el modelo con la sintaxis siguiente:

```
modelknn.fit(Xtrain,ytrain)
```

Para las métricas de clasificación:

```
from sklearn.metrics import confusion_matrix,
recall_score, precision_score
```

La sintaxis para las métricas es:

```
metricavar = metrica(ytrue,ypred)
```

Ver problema 1.