2023/2024 1Q

Lista de Problemas 2

APA

Javier Béjar

Departament de Ciències de la Computació

Grau en Enginyeria Informàtica - UPC

UPC FIB Facultat d'Informàtica de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

DEPARTAMENT DE CIÉNCIES DE LA COMPUTACIÓ

FACULTAT D'INFORMÁTICA DE BARCELONA

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Primera edición, septiembre 2021

Esta edición, Septiembre 2023

Instrucciones:

Para la entrega de grupo debéis elegir un problema del capítulo de problemas de grupo.

Para la entrega individual debéis elegir un problema del capítulo de problemas individuales.

Cada miembro del grupo debe elegir un problema individual diferente.

Debéis hacer la entrega subiendo la solución al racó.

Evaluación:

La nota de esta entrega se calculará como 1/3 de la nota del problema de grupo más 2/3 de la nota del problema individual.

Al realizar el informe correspondiente a los problemas explicad los resultados y las respuestas a las preguntas de la manera que os parezca necesaria. Se valorará más que uséis gráficas u otros elementos para ser más ilustrativos.

La parte que no es de programación la podéis hacer a mano y escanearla a un archivo **PDF**. Comprobad que **sea legible**.

Para la parte de programación podéis entregar los resultados como un notebook (Colab/Jupyter). Alternativamente, podéis hacer un documento explicando los resultados como un PDF y un archivo python con el código

También, si queréis, podéis poner las respuestas a las preguntas en el notebook, este os permite insertar texto en markdown y en latex.

Aseguraos de que los notebooks mantienen la solución que habéis obtenido, no los entreguéis sin ejecutar.

Objetivos:

1. Identificar y aplicar el preproceso más adecuado a un conjunto de datos
2. Saber aplicar técnicas de reducción de dimensionalidad para la visualización de datos
3. Saber crear y aplicar clasificadores generativos y discriminativos
4. Interpretar los resultados de un problema de clasificación

CAPÍTULO 1

Problemas de Grupo

1. **Datos muy desequilibrados**

   En los dominios médicos, los conjuntos de datos suelen tener un gran desequilibrio que dificulta la obtención de un modelo con buena precisión que realmente discrimine las clases que nos interesan. Vamos a trabajar con el conjunto de datos HCV del repositorio de conjuntos de datos de UCI que recopila medidas de muestras de sangre que corresponden a individuos sanos y pacientes con hepatitis C (https://archive.ics.uci.edu/ml/datasets/HCV+data). Leed primero la documentación del conjunto de datos.

   El objetivo de este problema es construir un clasificador que prediga si el ejemplo corresponde a una de las cuatro clases que tenemos (persona sana (Donante de Sangre) o una de las otras tres clases). Esta es una tarea más difícil que el objetivo original del conjunto de datos.

   a) El primer paso es preprocesar y preparar los datos antes de ajustar cualquier modelo. Primero debéis fusionar los ejemplos de todas las clases etiquetadas BloodDonor y ocuparos de los valores perdidos que existen en el conjunto de datos (imputarlos usando la media del atributo es suficiente). Descartad cualquier atributo que no sea informativo. Dividid los datos en conjuntos de entrenamiento y test[1] (70 %/30 %) y estandarizad las variables (calculad el estandarizador a partir de los datos de entrenamiento y luego aplicadlo a los datos de test).

   b) Aplicad algún método de reducción de dimensionalidad a los datos de entrenamiento y comentad lo que se pueda apreciar en la visualización. Pensad en qué podéis representar sobre la transformación.

   c) Nuestro modelo base habitual para la clasificación será Naïve Bayes (Gaussiano). Calculad el acierto de validación cruzada, mirad el informe de clasificación sobre el test y representad su matriz de confusión.

   d) Ajustad un discriminante lineal (LDA) y una regresión logística a los datos. Explorad los hiperparámetros de los modelos que los tengan ¿Son estos modelos mejores que Naïve Bayes? No os limitéis a comparar sus aciertos, ¿son mejores sus resultados en las clases minoritarias?

[1] Fijad el parámetro `random_state` en la función `train_test_split` para que los resultados no cambien cuando repita el experimento y haced la división estratificada.

e) La minimización de riesgos vecinales (Vicinal Risk Minimization) es un enfoque alternativo a la minimización del riesgo (en clase hablamos sobre la minimización empírica del riesgo) que establece que podemos abordar el aprendizaje utilizando una estimación de la densidad alrededor de los ejemplos. Esto puede verse como una técnica de aumento de datos (básicamente inventamos datos). Suponiendo que la densidad alrededor de un ejemplo es gaussiana, podemos añadir nuevos ejemplos a nuestro conjunto de datos agregándoles ruido gaussiano.

- Muestread de los datos de entrenamiento con reemplazo 500 y 1000 ejemplos de las tres clases minoritarias y agregad a cada uno ruido gaussiano con varianza 0.1 y 0.01 (tendréis un total de cuatro conjuntos de datos)
- Ajustad un discriminante lineal (LDA) y una regresión logística a estos conjuntos de datos

¿Ha habido alguna mejora en los resultados? ¿Cuál creéis que es el efecto de estos datos adicionales en los clasificadores? Pensad en lo que calculan los modelos y lo que está cambiando al agregar estos nuevos datos.

f) Una alternativa al aumento de datos para algunos clasificadores es asignar diferentes pesos a las clases o ejemplos. El modelo de regresión logística tiene un parámetro `class_weight` donde una de las opciones es `balanced`. Eso ponderará las clases según sus frecuencias. Ajustad una regresión logística con este parámetro y comparad los resultados con los anteriores.

2. **Decisiones difíciles**

En conjuntos de datos médicos es fundamental establecer cómo el modelo realiza las decisiones para que la clase de interés tenga el mejor rendimiento incluso cuando estemos cometiendo más errores en la clase menos importante. El conjunto de datos Indian Liver Patient Dataset del repositorio de UCI registra información de pacientes de hígado e individuos sanos (https://archive.ics.uci.edu/ml/datasets/ILPD+%28Indian+Liver+Patient+Dataset%29). Leed primero la documentación del conjunto de datos.

El objetivo de este problema es analizar diferentes clasificadores que predicen si el ejemplo corresponde a un paciente sano o no y determinar cuál/es permiten un mejor umbral de decisión para diagnosticar mejor la clase importante.

a) El primer paso es preprocesar y preparar los datos antes de ajustar cualquier modelo. Debéis ocuparos de los valores faltantes que existen en el conjunto de datos (imputarlos utilizando la media del atributo es suficiente). Dividid los datos en conjuntos de entrenamiento y test[2] (70 %/30 %) y estandarizad las variables (calculad el estandarizador a partir de los datos del entrenamiento y luego aplicadlo a los datos de test).

b) Aplicad algún método de reducción de dimensionalidad a los datos de entrenamiento y comentad lo que se pueda apreciar en la visualización. Pensad en qué podéis representar sobre la transformación.

c) Nuestro modelo base habitual para la clasificación será Naive Bayes (Gaussiano). Calculad el acierto de validación cruzada, mirad el informe de clasificación sobre el test y representad su matriz de confusión. Representad la curva ROC[3] usando como referencia la clase que más importa y observad cómo aumentan los falsos positivos a medida que cambia el

[2] Fijad el parámetro `random_state` en la función `train_test_split` para que los resultados no cambien cuando repitáis el experimento y haced la división estratificada.

[3] La curva Receiver Operating Characteristic representa la proporción de verdaderos positivos frente a la proporción de falsos positivos para diferentes umbrales de decisión del modelo.

umbral de decisión. La función `roc_curve` os dará los datos representados en el gráfico y los umbrales de decisión. Pensad en las implicaciones de cada umbral y elegid uno que parezca razonable para el problema (verdaderos positivos altos y falsos positivos no muy altos). Fijaos que ese umbral determina como se etiquetan los ejemplos, obtened las probabilidades de cada ejemplo para el test y asignad de nuevo las predicciones a los ejemplos. Usad estas nuevas etiquetas para calcular el acierto y la precisión y recuperación para cada clase (buscad las funciones que las calculan en scikit-learn).

d) Ajustad un modelo discriminante lineal (LDA) a los datos y comparad los resultados con Naïve Bayes. ¿Existe un umbral de decisión que mejore el clasificador Naïve Bayes? Recalculad las predicciones con ese umbral y comparad resultados.

e) Ajustad un modelo de regresión logística a los datos y comparad los resultados con los de Naïve. ¿Existe un umbral de decisión que mejore con respecto al clasificador Naïve Bayes? Recalculad las predicciones con ese umbral y comparad resultados.

f) Tal vez el límite de decisión entre las dos clases es lo suficientemente complejo como para que los clasificadores lineales no puedan obtener buenos resultados. Ajustad un modelo de k-vecinos más cercanos (pensad en cómo se deben preprocesar los atributos para este modelo) explorando sus hiperparámetros. ¿Tiene este modelo mejor rendimiento? Decidid un umbral y recalculad las predicciones. Comparad resultados. ¿Qué modelo elegiríais?

3. **Aprendiendo a sumar**

Aprender a sumar dos números parece un problema fácil, pero es difícil enseñárselo a una máquina a partir de ejemplos. El objetivo de este problema es probar qué tan bueno es k-vecinos más cercanos aprendiendo esta tarea.

Vamos a limitar la tarea a aprender a sumar dos números de tres dígitos, que pueden dar como resultado un número de cuatro dígitos como máximo. Esto se puede definir como un problema de clasificación de salida múltiple. Esto significa que el clasificador devuelve una predicción que corresponde a un conjunto de clases en lugar de una sola clase.

Vamos a definir el problema como ejemplos que tienen seis atributos, los primeros tres atributos corresponden a los tres dígitos del primer número y los últimos tres a los dígitos del segundo número. La salida tiene cuatro valores, uno para cada dígito del número del resultado de la suma. Por ejemplo:

X1	X2	X3	X4	X5	X6	Y1	Y2	Y3	Y4
0	1	6	4	7	7	0	4	9	3

a) La primera tarea es implementar una función que genera un conjunto de datos aleatorios de sumas de dos números de tres dígitos. Aseguraos de que no haya ejemplos repetidos en el conjunto de datos.

b) Generad conjuntos de datos de tamaño creciente de hasta el 20 % de los ejemplos posibles en este problema. Notad que hay 10^6 ejemplos posibles. Dividid los conjuntos de datos en un conjunto de entrenamiento y test con una proporción de 90 %/10 %

c) Aplicad algún método de reducción de dimensionalidad a alguno de los conjuntos de datos de entrenamiento y comentad lo que se pueda apreciar en la visualización. Pensad en qué podéis representar sobre la transformación.

d) Para poder trabajar con la clasificación de salida múltiple, debéis usar el objeto `MultiOutputClassifier` de `scikit learn`. Esto envolverá el modelo clasificador k vecinos más cercanos para que pueda realizar este tipo de clasificación.

Tenéis básicamente tres hiperparámetros que podéis explorar. El número de vecinos (no tiene que ser exhaustivo, pero aseguraos de probar diferentes posibilidades y no seáis tímidos con el número de vecinos), la distancia utilizada para calcular los vecinos (encontraréis diferentes distancias disponibles en la documentación de K-nn) y si se utiliza la distancia para promediar las decisiones de los vecinos. Ajustad estos hiperparámetros para encontrar el clasificador K-nn con el mejor rendimiento para cada conjunto de datos.

Llevará demasiado tiempo realizar la validación cruzada, así que ajustad los parámetros utilizando la puntuación obtenida por el conjunto de test. Utilizad también el parámetro `n_jobs=-1` para la clasificación K-nn y multisalida, de modo que pueda aprovechar el cálculo multinúcleo[4].

Explicad las cosas que habéis intentado y los resultados, no deis solo los mejores resultados. Tratad de dar una explicación sobre por qué algunas opciones tienen un mejor rendimiento que otras.

e) A veces usar una representación diferente para los atributos facilita la tarea de aprendizaje. Dado que en realidad tenemos atributos categóricos que corresponden a los dígitos de los números, podemos codificarlos utilizando una codificación one hot. Transformad los atributos del conjunto de datos y calculad la precisión de los mejores hiperparámetros para cada conjunto de datos y comparad su precisión con los resultados anteriores. Representad la precisión del mejor clasificador para cada representación y tamaño de conjunto de datos. Comentad los resultados e intentad explicar lo que ha sucedido.

f) ¿Por qué creéis que esta tarea es difícil de aprender? No os limitéis a decir que hay un acarreo en la suma, ¿que implica eso respecto a que asumimos en Knn al usar distancias? ¿Sería más fácil si en lugar de resolverlo como un problema de clasificación de múltiples salidas usamos la suma como la clase para predecir?

[4] Si usáis Colab para este problema solo dispondréis de 2 cores, si lo ejecutáis en cualquier otra máquina que tenga más cores os irá más rápido-

CAPÍTULO 2

Problemas Individuales

Para obtener los datos para estos problemas necesitaréis instalaros la última versión de la libreria `apafib`. La podéis instalar localmente haciendo:

```
pip install -user -upgrade apafib
```

Para usar las funciones de carga de datos solo tenéis que añadir su importación desde la librería, en vuestro script o notebook, por ejemplo

```
from apafib import load_literature
```

La función os retornara un `DataFrame` de Pandas o arrays de numpy con los datos y la variable a predecir, cada problema os indicará que es lo que obtendréis.

1. **La evolución de la literatura**

   Esta claro que la literatura va cambiando con el tiempo, tanto en los temas como en su vocabulario. Esto debería poder ayudar a datar un fragmento de texto dado el vocabulario que se utiliza. En este problema vamos a usar fragmentos extraídos al azar de obras de autores desde el siglo XVI al XX con el objeto de comprobar si realmente se puede realizar esa clasificación.

   Trabajaremos con un conjunto de datos compuesto por fragmentos de obras de 23 autores seleccionados[1] de esos siglos que podéis obtener mediante la función `load_literature` de la librería `apafib`. Esta función retornará una lista con los textos y otra con las etiquetas que les corresponden. Resuelve los siguientes apartados ilustrando los resultados de la manera que te parezca más adecuada.

   a) El trabajar con texto es algo diferente de los datos tabulares habituales. En este tipo de datos se obtiene la matriz de datos a partir de extraer un subconjunto de las palabras de los textos y hacer un cálculo sobre ellas como por ejemplo determinar si están o no en un documento o el número de veces que aparece. Esto lo podemos hacer con la función `CountVectorizer` del scikit-learn que es precisamente para este tipo de datos. Generaremos matrices de datos variando el tamaño del vocabulario, usaremos 250 y 500 palabras[2].

[1] Los textos se han extraído de libros mantenidos por el proyecto Gutemberg.

[2] Fijate que este modelo tiene un parámetro `max_features` que te permite escoger el tamaño del vocabulario.

Podemos generar la matriz de datos para que contenga solo si las palabras aparecen o no en cada ejemplo o el número de veces que aparecen, usaremos las dos opciones. Utiliza el parámetro `stop_words` que tiene esta función indicando que el idioma del texto es el inglés. Esto nos dará cuatro conjuntos de datos diferentes. Fíjate que esto **es un preproceso** como otros que hemos ido empleando, así que aplícalo correctamente a los datos. Genera una partición de entrenamiento y una de test (80 %/20 %) que sea estratificada (fija también el estado del generador de números aleatorios para la reproducibilidad).

b) Los datos claramente no cumplen los supuestos de PCA, asi que podemos intentar visualizarlos mediante t-SNE. ¿Se puede ver alguna separabildad entre los datos en la proyección en 2D? Comenta los resultados.

c) Tenemos datos que o son binarios (Bernoulli) o corresponden a distribuciones multinomiales respectivamente. Si asumimos que las palabras de un texto son independientes (no lo son realmente) podemos usar Naïve Bayes para clasificarlos. Tienes en scikit-learn modelos de Naïve Bayes que usan estas dos distribuciones. Aplica estos métodos a los conjuntos de datos que has generado y determina la calidad de los modelos (la calidad no se limita solo al acierto). Comenta los resultados.

d) Los modelos generativos funcionan bien si la suposición que hacemos sobre la distribución de los datos es correcta. Los modelos discriminativos como Regresión Logística pueden ser adecuados cuando no esta clara la distribución de los atributos. Ajusta un modelo de regresión logística a los conjuntos de datos que has generado explorando los hiperparámetros del modelo y determina la calidad de los mejores modelos. Comenta los resultados. ¿que modelo elegirías entre todos? ¿Crees que si vamos aumentando el número de palabras que usamos en el modelo esté irá mejorando en calidad?

2. **Original o traducción**

A veces nos interesa saber si un texto corresponde a la fuente original o ha sido derivado de ella de alguna manera. En este problema intentaremos averiguar si un texto proviene de la lengua original (Inglés) o es una traducción. Vamos a usar fragmentos extraídos al azar de obras de autores de diferentes épocas y géneros con el objeto de comprobar si realmente se puede realizar esa clasificación. Estos están escritos en inglés, pero algunos de ellos han sido traducidos de otros idiomas, como el ruso, el francés o el italiano entre otros.

Trabajaremos con un conjunto de datos compuesto por fragmentos de obras de 20 autores seleccionados[3] que podéis obtener mediante la función `load_translation` de la librería `apafib`. Esta función retornará una lista con los textos y otra con las etiquetas que les corresponden. Resuelve los siguientes apartados ilustrando los resultados de la manera que te parezca más adecuada.

a) El trabajar con texto es algo diferente de los datos tabulares habituales. En este tipo de datos se obtiene la matriz de datos a partir de extraer un subconjunto de las palabras de los textos y hacer un cálculo sobre ellas como por ejemplo determinar si están o no en un ejemplo o el número de veces que aparece. Esto lo podemos hacer con la función `CountVectorizer` del scikit-learn que es precisamente para este tipo de datos. Generaremos matrices de datos variando el tamaño del vocabulario, usaremos 2500 y 5000 palabras[4], ya que nos imaginamos que decidir en este problema estará en los detalles. Podemos generar la matriz de datos para que contenga solo si las palabras aparecen o no en cada ejemplo o el número de veces que aparecen, utilizaremos las dos opciones. Emplea el parámetro `stop_words` que tiene esta función indicando que el idioma del texto es el inglés. Esto nos dará cuatro

[3] Los textos se han extraído de libros mantenidos por el proyecto Gutemberg.

[4] Fijate que este modelo tiene un parámetro `max_features` que te permite escoger el tamaño del vocabulario.

conjuntos de datos diferentes. Fíjate que esto **es un preproceso** como otros que hemos ido empleando, así que aplícalo correctamente a los datos. Genera una partición de entrenamiento y una de test (80 %/20 %) que sea estratificada (fija también el estado del generador de números aleatorios para la reproducibilidad).

b) Los datos claramente no cumplen los supuestos de PCA, así que podemos intentar visualizarlos mediante t-SNE. ¿Se puede ver alguna separabilidad entre los datos en la proyección en 2D? Comenta los resultados.

c) Tenemos datos que o son binarios (Bernoulli) o corresponden a distribuciones multinomiales respectivamente. Si asumimos que las palabras de un texto son independientes (no lo son realmente) podemos usar Naïve Bayes para clasificarlos. Tienes en scikit-learn modelos de Naïve Bayes que emplean estas dos distribuciones. Aplica estos métodos a los conjuntos de datos que has generado y determina la calidad de los modelos. Comenta los resultados.

d) Los modelos lineales funcionan bien si existe una buena separación entre las clases, pero la frontera puede ser suficientemente compleja para que no podamos superar cierto límite. El modelo de k vecinos cercanos si puede aproximar fronteras complejas y parece adecuado para este problema. Ajusta un modelo de Knn a los conjuntos de datos que has generado explorando sus hiperparámetros, y determina la calidad de los mejores modelos. Piensa también si es adecuado el normalizar de alguna manera los datos. Comenta los resultados. Vistos los resultados obtenidos por los clasificadores y teniendo en cuenta lo que hace falta para realizar inferencia con cada uno de ellos, ¿qué modelo de entre todos los que has obtenido elegirías? Razona la respuesta.

3. **Eso es un siete o un nueve, ¿o quizás un cuatro?**

   Parece haber una obsesión con conjuntos de datos que representan dígitos escritos a mano. Esto tiene una explicación sencilla. Uno de los primeros retos del OCR fue el reconocimiento de códigos postales en cartas. El clasificar cartas a mano consume demasiado tiempo y no es completamente fiable (además de ser muy aburrido), por lo que tener un sistema eficiente para clasificar el correo era un problema interesante[5]. Uno de los conjuntos de datos más conocido es MNIST digits. Vamos a utilizar una parte de este conjunto de datos, en concreto un subconjunto de los dígitos 4, 7 y 9, para experimentar con varios clasificadores. Podéis obtenerlos mediante la función `load_MNIST` de la librería `apafib`. Esta función retornará cuatro matrices de datos, el conjunto de entrenamiento y sus etiquetas y el conjunto de test y sus etiquetas. Resuelve los siguientes apartados ilustrando los resultados de la manera que te parezca más adecuada.

   a) Los datos corresponden a imágenes de 28×28 píxeles en niveles de gris. Estos ya están convertidos a vectores y normalizados a la escala [0,1]. Aplica PCA a los datos de entrenamiento y represéntalos en 2D. ¿Se puede ver separabilidad entre las clases? Comenta el resulado

   b) Asumiendo que los píxeles son independientes y se distribuyen de manera gausiana, ajusta un modelo Naïve Bayes gausiano y evalúa la calidad del modelo. Cuanto más acerquemos los datos a lo que supone el modelo que ajustamos probablemente obtengamos un mejor resultado. Una posibilidad es binarizar la matriz de datos usando un límite de 0.5 y asumir que los píxeles siguen una distribución de Bernoulli. Transforma los datos a binarios utilizando la función `Binarizer` y ajusta un Naïve Bayes Bernoulli. Evalúa la calidad del modelo y comenta los resultados.

[5] Ahora ya nadie escribe cartas, pero siempre habrá texto escrito a mano que reconocer en otras aplicaciones.

c) En esta aplicación la calidad del modelo es bastante importante y es probable que las fronteras entre los dígitos sean complejas. Ajusta un modelo k nearest neighbours a la matriz original y la binarizada explorando adecuadamente los hiperparámetros de este modelo. Comenta los resultados

d) Habás observado al hacer el PCA que el número de componentes necesario para explicar la variancia de los datos no es muy grande respecto a su dimensionalidad real. Elige un número de componentes, obtén las matrices de datos transformadas. Ajusta un modelo Naïve Bayes y un K nearest neighbours a estos datos explorando adecuadamente sus hiperparámetros. Evalúa la calidad de los modelos. ¿Por qué sería más correcto usar el Naïve Bayes con esta transformación? ¿Qué ventajas/inconvenientes tendría el trabajar con los datos transformados? ¿Qué modelo de entre todos elegirías para reconocer códigos postales? Razona tu respuesta.

4. **Se discreto**

Hay muchos preprocesos que se pueden aplicar a los atributos de un conjunto de datos. Estamos acostumbrados a trabajar con datos continuos, pero a veces es más fácil cruzar al mundo discreto por diferentes motivos. En este problema vamos a experimentar con diferentes discretizaciones.

El conjunto de datos *apendicitis* está incluido en los Penn Machine Learning Benchmarks. Para descargarlo tendrás que seguir las instrucciones de su página web (`https://epistasislab.github.io/pmlb/index.html`).

a) Vamos a comenzar con el conjunto de datos original. Divide los datos en entrenamiento y test (60 %/40 %), la partición debe estratificarse (fija también el estado del generador de números aleatorios para la reproducibilidad).

b) Entrena un naive bayes, una regresión logística y un K-vecinos más cercanos explorando sus hiperparámetros como creas conveniente. Preprocesa primero los datos adecuadamente para estos modelos. Obten el acierto de validación cruzada, el acierto en test, la matriz de confusión y el informe de clasificación.

c) Scikit-learn tiene un preproceso capaz de discretizar datos continuos (`KBinsDiscretizer`). Este preproceso puede convertir cada variable a un conjunto de valores discretos de diferentes maneras. Los métodos más adecuados son los que tratan de aproximar de alguna manera la distribución de los datos originales. La discretización `quantile` usa los cuantiles de la distribución asumiendo gausianidad. La discretización `kmeans` no asume una distribución específica y busca un número de densidades compactas en los datos. Vamos a aplicar ambos métodos a todos los atributos para obtener una discretización en 2, 3 y 4 valores (usa `onehot-dense` como codificación para que cada valor sea ahora un atributo del conjunto de datos). Fíjate que esto **es un preproceso** como otros que hemos ido empleando, así que aplícalo correctamente a los datos. Ajusta los mismos modelos que antes teniendo en cuenta que ahora tienes variables binarias y compara los diferentes resultados. ¿ha afectado este proceso a la calidad de los modelos? ¿Cuál elegirías y por qué?

d) Teniendo en cuenta la interpretabilidad de los modelos, ¿crees que tiene ventajas el trabajar con datos discretizados? ¿Por qué?

5. **Problemas de corazón**

Los problema de corazón son algo serio por lo que es importante el poder predecirlos con antelación. El conjunto de datos Heart Failure clinical records del que os podeis documentar en este enlace `https://archive.ics.uci.edu/ml/datasets/Heart+failure+clinical+records`, registra información clínica sobre pacientes y si fallecieron o no durante el periodo de seguimiento (columna `DEATH_EVENT`). Podéis obtener estos datos mediante la función `load_heart_failure`

de la librería `apafib`. Esta función retornará la matriz de datos. Resuelve los siguientes apartados ilustrando los resultados de la manera que te parezca más adecuada.

a) Divide los datos en un conjunto de entrenamiento y otro de test (70 %/30 %), la partición debe estratificarse (fija también el estado del generador de números aleatorios para la reproducibilidad). Haz una visualización mínima de las variables para hacerte una idea de los datos y calcula sus correlaciones. Verás que hay variables continuas y discretas. Algunas de las continuas no parecen muy gausianas. Haremos dos conjuntos de datos, uno que tenga las variables originales y otro en el que las variables que se desvíen bastante de la gausianidad[6] están transformadas con la función logaritmo. Normaliza adecuadamente los datos.

b) Empezaremos por Naive Bayes, ajusta un modelo para distribuciones gausianas y evalúa adecuadamente la calidad de los modelos.

c) Al hacer la exploración de los datos probablemente viste que hay cierta correlación entre variables, por lo que es posible que los datos se desvíen de lo que asume Naïve Bayes. Ajusta un modelo discriminante lineal (LDA) a los dos conjuntos de datos y evalúa adecuadamente la calidad de los modelos.

d) Tenemos un conjunto de datos que mezcla variables discretas y continuas, por lo que es posible que los modelos generativos no sean completamente adecuados. Ajusta un modelo de regresión logística a los dos conjuntos de datos explorando los hiper parámetros de este modelo adecuadamente (el conjunto de datos es pequeño, asi que el ajuste debería ser rápido). Evalúa la calidad de los modelos.

Dado que las clases que tiene el conjunto de datos no están balanceadas y de hecho nos importa más una de las clases podríamos usar el parámetro `class_weight` de este modelo para que se tenga en cuenta esta circunstancia usando del valor `balanced`. Haz un nuevo ajuste de la regresión logística con los dos conjuntos de datos utilizando este hiperparámetro. ¿qué modelo elegirías de entre todos los que has obtenido? ¿Por qué?

6. **Ponte en forma**

Muchos de los modelos que usamos asumen que los datos son gausianos y son tolerantes a alguna desviación de esa suposición, pero usar datos con la distribución incorrecta generalmente es perjudicial para el rendimiento del modelo. Afortunadamente, hay ciertas transformaciones que se pueden aplicar a los datos que pueden acercar muchas distribuciones extrañas a la forma gaussiana[7]. En este problema vamos a experimentar con algunas de estas transformaciones.

El conjunto de datos *glass2* está incluido en los Penn Machine Learning Benchmarks[8]. Para descargarlo tendrás que seguir las instrucciones de su página web (https://epistasislab.github.io/pmlb/python-ref.html).

a) Vamos a comenzar con el conjunto de datos original. Divide los datos en entrenamiento y test (60 %/40 %), la partición debe estratificarse (fija también el estado del generador de números aleatorios para la reproducibilidad). Estandariza las variables y representa sus histogramas para verificar qué tan gausianas se ven. Usa un test de gausianidad para saber cuanto se desvían de esa distribución.

[6] Queremos que las distribuciones estén más o menos centradas

[7] Estas transformaciones vienen con una advertencia, no son lineales, por lo que pueden afectar la relación que podría tener una variable con otras, aunque funcionan bien para variables independientes, esa también es una suposición habitual de nuestros modelos.

[8] https://epistasislab.github.io/pmlb/index.html

b) Entrena un naïve bayes, una regresión logística y un K-vecinos más cercanos explorando sus hiperparámetros como creas conveniente. Obtén el acierto de validación cruzada, el acierto en el test, la matriz de confusión y el informe de clasificación.

c) Scikit-learn tiene dos métodos para transformar los datos de su distribución original a algo más cercano a gausiano `QuantileTransformer` y `PowerTransformer`. El primero usa los cuantiles de los datos para que coincidan con los cuantiles de la gausiana, básicamente hace corresponder la CDF empírica de los datos y la CDF teórica de la gausiana. La segunda permite aplicar dos transformaciones no lineales a los datos la *Yeo-Johnson* y la *Box-Cox*[9].

Aplica estas transformaciones a todas las variables de los datos para obtener tres conjuntos de datos diferentes (todavía tendrás que estandarizar el resultado) y ajusta los mismos modelos que antes. ¿Ha afectado este proceso a la calidad de los modelos? ¿Cuál elegirías y por qué?

d) Hemos aplicado las transformaciones a todas las variables, pero probablemente algunas de ellas ya eran más o menos gausianas. Selecciona las variables menos gausianas, utiliza la transformación que mejores resultados haya obtenido y ajusta el modelo que hayas elegido como el mejor. ¿Ha mejorado el modelo? Calcula las correlaciones entre las variables que has seleccionado y el resto de variables antes y después de la transformación. ¿Hay alguna diferencia? En el caso de que haya una diferencia, ¿qué impacto tiene en como se modelan los datos?

[9]Esta transformación tiene la restricción de que solo aplica a datos positivos, para el conjunto de datos específico de este problema no hay datos negativos, pero hay ceros, agregar un épsilon a estos valores será suficiente.