



Desafío - Máquinas de Soporte Vectorial

- Para realizar este desafío debes haber revisado la lectura y videos correspondiente a la unidad.
- Crea una carpeta de trabajo y guarda todos los archivos correspondientes (notebook y csv).
- Una vez terminado el desafío, comprime la carpeta y sube el `.zip` a la sección correspondiente.

Descripción

Para esta sesión trabajaremos con la base de datos sobre cáncer mamario de Wisconsin. El objetivo es desarrollar un Clasificador mediante Máquinas de Soporte de Vectores que predica de forma adecuada en base a una serie de atributos sobre la composición del núcleo de una célula mamaria. Para más detalles técnicos asociados a la base de datos, pueden hacer click en el [link](#).

Ejercicio 1: Preparar el ambiente de trabajo

- Importe todas las librerías a utilizar.
- Fije los parámetros de los gráficos con `plt.rcParams`.
- Excluya las columnas `id` y `Unnamed: 32` de la base de datos.
- Recodifique el vector objetivo `diagnosis` a numérico para poder procesarlo posteriormente.

Ejercicio 2: Visualizando la distribución de los atributos

- Para cada uno de los atributos, grafique los histogramas condicional a cada clase del vector objetivo.
- Agregue las medias correspondientes y reporte a grandes rasgos cuáles son los atributos con una mayor similitud en la distribución.

Ejercicio 3: Estimando el porcentaje de overlap en los atributos

- Parte de las virtudes de las Máquinas de Soporte Vectorial es la capacidad de lidiar con clases no separables mediante el proceso de kernelización. Resulta que un aspecto importante que muchas veces se obvia es medir la noseparabilidad de los atributos, condicional a cada clase del vector objetivo.
- El procedimiento para estimar el rango de noseparabilidad entre clases se implementa en Python de la siguiente manera:

```
def histogram_overlap(df, attribute, target, perc=100):  
    # get lower bound  
    empirical_lower_bound = np.floor(df[attribute].min())  
    # get upper bound  
    empirical_upper_bound = np.ceil(df[attribute].max())  
    # preserve histograms  
    tmp_hist_holder = dict()  
  
    # for each target class  
    for unique_value in np.unique(df[target]):  
        # get histogram  
        tmp, _ = np.histogram(  
            # for a specific attribute  
            df[df[target] == unique_value][attribute],  
            # define percentage  
            bins=perc,  
            # limit empirical range for comparison  
            range=[empirical_lower_bound, empirical_upper_bound]  
        )  
        # append to dict  
        tmp_hist_holder["h_"+str(unique_value)] = tmp  
  
    get_minima = np.minimum(tmp_hist_holder["h_1"], tmp_hist_holder["h_-1"])  
    intersection = np.true_divide(np.sum(get_minima),  
    np.sum(tmp_hist_holder["h_-1"]))  
    return intersection
```

- La intersección devolverá el porcentaje de comunalidad entre ambas clases, donde mayores niveles indican una mayor comunalidad.
- Utilizando la función, genere un dataframe donde almacenará el nombre del atributo y su porcentaje. Ordene este dataframe de forma descendente y preserve.

Ejercicio 4: Selección del modelo por GridSearchCV

- Entrene una serie de modelos `SVC` con los siguientes hiperparámetros:
 - `C: [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]`.
 - `gamma: [0.0000001, 0.0001, 0.001, 0.01, 0.1, 1, 10]`.
 - Validaciones cruzadas: 10.
- Genere un heatmap en base a los puntajes estimados con `GridSearchCV`.

tip: Vea cómo acceder a la llave `mean_test_score` en el diccionario `cv_results_`.

Digresión: Un par de elementos a considerar en la implementación de `GridSearchCV`

Si trabajamos con `sklearn.model_selection.GridSearchCV`, tan solo haciendo la división en dos muestras es suficiente, incorporando los conjuntos `X_train` y `y_train` a nuestro objeto instanciado y preservando `X_test` e `y_test` como una muestra de validación externa. Si tenemos un archivo de testing externo, se recomienda no hacer división.

- El objeto creado con `sklearn.model_selection.GridSearchCV` sigue la misma funcionalidad de cualquier método de estimación de `scikit-learn`, con los pasos de Instanciar y Entrenar. Este objeto tendrá muchos elementos a considerar:
 - `sklearn.model_selection.GridSearchCV.cv_results_` devolverá un diccionario donde las llaves representarán distintas métricas y los valores representarán el desempeño de cada modelo:
 - `split*` Indicará la métrica específica en cada validación cruzada y combinación de hiperparámetros.
 - `*_time` Indicará el tiempo de ejecución en cada modelo.
 - Por lo general trabajaremos con `mean_test_score` y `mean_train_score` que representa la media de CV para cada combinación de hiperparámetros.
 - `sklearn.model_selection.GridSearchCV.best_estimator_` devuelve un modelo listo para entrenar con la mejor combinación de hiperparámetros.
 - `sklearn.model_selection.GridSearchCV.best_score_` devuelve el desempeño promedio del modelo en el testing interno. Si es un problema de clasificación devolverá `Accuracy`, si es un problema de regresión devolverá `MSE`.
- Reporte en qué rango de cada hiperparámetro el modelo presenta un desempeño eficiente. Reporte la mejor combinación de hiperparámetros y el desempeño en la muestra de entrenamiento.

Ejercicio 5: Validación del modelo en el hold-out sample

- Genere las predicciones del hold-out sample en base a la mejor combinación de hiperparámetros. Genere un reporte con las métricas de desempeño clásicas para los modelos de clasificación. Comente en qué casos el modelo presenta un desempeño deficiente

Ejercicio (opcional): Depuración de atributos

- Reentrene el modelo en función de los atributos que presenten un coeficiente de overlap menor a .45.
- Reporte el desempeño del modelo y comente sobre los nuevos hiperparámetros estimados, así como su desempeño en comparación al modelo del ejercicio 5.