Sustentación Tópicos y Aprendizajes de Máquinas

Taller I

February 23, 2024

1 Recreación Cross Validation

En el experimento del libro Elements of Statistical Learning 7.10.2 The Wrong and Right Way to Do Cross-validation.Nos pide considerar un problema de clasificación con un gran número de predictores, como podría surgir, por ejemplo, en aplicaciones genómicas o proteómicas. Una estrategia típica para el análisis podría ser la siguiente:

1. Filtrar los predictores: encontrar un subconjunto de predictores "buenos" que muestren una correlación bastante fuerte (univariada) con las etiquetas de clase.
2. Utilizar solo este subconjunto de predictores para construir un clasificador multivariado.
3. Utilizar validación cruzada para estimar los parámetros de ajuste desconocidos y para estimar el error de predicción del modelo final.

Para lo anterior, nos pide considerar un escenario con $N = 50$ muestras en dos clases de tamaño igual, y $p = 5000$ predictores cuantitativos (distribuidos normal estándar) que son independientes de las etiquetas de clase. La tasa de error verdadera (de prueba) de cualquier clasificador es del 50%. Para lo cual, creamos la siguiente función en python que genera una base de datos síntetica gracias al *make_classification* de *scikit-learn* dados los parámetros de N y p, asimismo ajustamos 2 clases con el mismo peso y lo convertimos a un dataframe de *pandas* que sea más fácil de manejar:

```python
def create_synth_df(N, p):
    # Hacemos un dataset sint tico con 5000 predictores
    synthetic_dataset, labels = make_classification(
        n_samples=N,  # Total de muestras
        n_features=p,  # Numero de predictores
        n_classes=2,  # Numero de clases
        n_clusters_per_class=1,  # Numero de clusters por clase
        weights=[0.5, 0.5],  # Pesos de las clases
    )

    # Creamos un dataframe con el dataset sint tico
    df = pd.DataFrame(synthetic_dataset, columns=[f"predictor_{i}" for i in range(p)])
    # A adimos la columna de las etiquetas
    df['label'] = labels

    return df
```

Ahora bien, el experimento nos pide calcular los 100 predictores con la mayor correlación con las etiquetas de clase, que corresponde al primer paso. Para esto, creamos la función *get_top_predictors* que calcula la matrix de correlación dado un dataframe y calcula los n predictores con mayor correlación de la variable label:

```python
def get_top_predictors(df, n):
    # Calculamos la matriz de correlaci n
    correlation_matrix = df.corr()
    # Tomamos los 100 predictores con mayor correlaci n con la variable objetivo
    top_predictors = correlation_matrix['label'].nlargest(n).index.tolist()
    #Devolver los n predictores con mayor correlaci n
    return top_predictors
```

Posteriormente, hacemos 5-Fold Cross-validation con el class label usando un 1-nearest neighbor classifier como pide el experimento para lo cual hacemos dos funciones distintas: *kNN_top_cv_error* que lo calcula para el dataframe con los predictores de mayor correlación y *kNN_cv_error* que lo hace con el dataframe general:

```
def kNN_top_cv_error(df, top_predictors):
    # Tomamos los 100 predictores con mayor correlaci n con la variable objetivo
    X = df[top_predictors].values
    Y = df['label'].values

    # Creamos un clasificador kNN
    knn = KNeighborsClassifier(n_neighbors=1)
    # Calculamos el accuracy usando validaci n cruzada
    accuracy = cross_val_score(knn, X, Y, cv=5, scoring='accuracy').mean()
    #Calculamos el CV error
    cv_error = 1 - accuracy
    #Devuelve el error de validaci n cruzada
    return cv_error
```

```
def kNN_cv_error(df):
    #Tomamos el dataframe inicial
    X = df.drop('label', axis=1).values
    Y = df['label'].values
    #Creamos un clasificador kNN
    knn = KNeighborsClassifier(n_neighbors=1)
    #Calculamos el accuracy usando validaci n cruzada
    accuracy = cross_val_score(knn, X, Y, cv=5, scoring='accuracy').mean()
    #Calculamos el CV error
    cv_error = 1 - accuracy
    #Devuelve el error de validaci n cruzada
    return cv_error
```

Finalmente, hacemos 50 simulaciones y calculamos el cv error para ambos casos lo que nos debe dar un error al 3% lejos del 50% de todos los predictores. Esto lo hacemos con las siguientes líneas de código:

```
cv_top_errors = []
cv_errors = []

for i in range(50):
    df = create_synth_df(50, 5000)
    top_predictors = get_top_predictors(df, 100)

    cv_error = kNN_top_cv_error(df, top_predictors)
    cv_error2 = kNN_cv_error(df)

    cv_top_errors.append(cv_error)
    cv_errors.append(cv_error2)


print("CV Errors for top predictors:", cv_top_errors)
print("CV Errors for all predictors:", cv_errors)
```

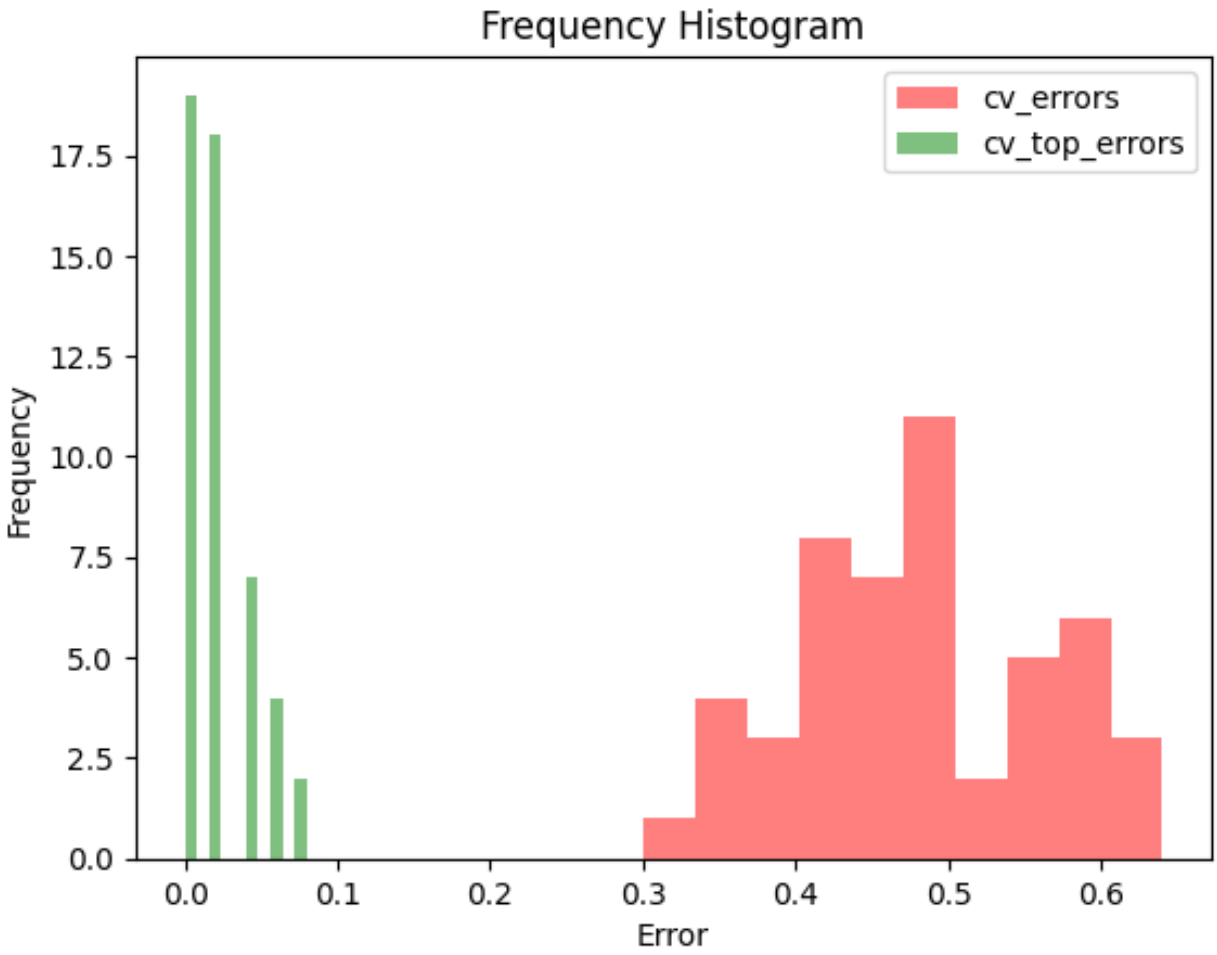

Figure 1: Frecuencia de CV Error

Al final, al igual que recalcaba el experimento obtuvimos resultados muy parecidos donde el dataframe con todos los predictores dió un error del 48.2% y el dataframe con los mejores cien predictores un error igual al 2.1%

2 Entrenamiento de Modelo

Antes de correr cualquier modelo es necesario obtener algunas estadísticas descriptivas de los valores. Además, encontramos que el dataframe no tiene missing values, tiene nueve features que son variables continuas, un target de clasificación binaria con 702 valores 0 y 298 valores iguales a 1 por lo cual el dataframe no se encuentra balanceado. Las estadísiticas descriptivas se muestran a continuación.

	feature_0	feature_1	feature_2	feature_3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	-0.004543	-0.032966	0.057683	0.007632
std	1.002503	1.145290	0.955562	1.009872
min	-4.032602	-5.051559	-3.437063	-4.267186
25%	-0.652377	-0.724177	-0.569093	-0.668221
50%	0.044962	-0.102816	0.045899	0.045645
75%	0.646045	0.553662	0.703013	0.668420
max	3.249901	4.594686	2.710780	2.799294

Table 1: Estadísticas básicas de las columnas numéricas (Parte 1)

	feature_4	feature_5	feature_6	feature_7
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.012276	0.021229	0.171337	-0.127766
std	0.980226	0.998634	1.540330	1.056942
min	-3.253993	-3.063942	-4.835950	-3.317522
25%	-0.633988	-0.683372	-0.888353	-0.742152
50%	0.016508	0.038008	0.042526	-0.255887
75%	0.686445	0.715232	1.174232	0.528072
max	3.027182	3.121180	6.762730	3.678459

Table 2: Estadísticas básicas de las columnas numéricas (Parte 2)

	feature_8	feature_9	target
count	1000.000000	1000.000000	1000.000000
mean	-0.003169	0.198735	0.298000
std	1.071977	1.000881	0.457608
min	-4.814347	-3.499941	0.000000
25%	-0.694583	-0.380670	0.000000
50%	0.086531	0.386678	0.000000
75%	0.691234	0.745101	1.000000
max	3.331401	3.375467	1.000000

Table 3: Estadísticas básicas de las columnas numéricas (Parte 3)

Las tablas presentan estadísticas descriptivas de las características numéricas de un conjunto de datos. En la primera tabla, se observa que las características 0 a 3 tienen valores medios cercanos a cero y una dispersión moderada, con valores que van desde -4 hasta 3.25 en el caso del mínimo y máximo, respectivamente. En la segunda tabla, las características 4 a 7 muestran una dispersión similar, con valores medios alrededor de 0 y una variabilidad similar a la primera tabla. En la tercera tabla, las características 8 y 9 presentan valores medios cercanos a cero y una dispersión comparable a las anteriores, junto con estadísticas para la variable objetivo, donde se observa que el 30% de los valores son 1. Estas tablas brindan una visión detallada de la distribución de las características numéricas y la variable objetivo en el conjunto de datos. También hicimos los histogramos de los datos los cuales no nos dieron mucha información:

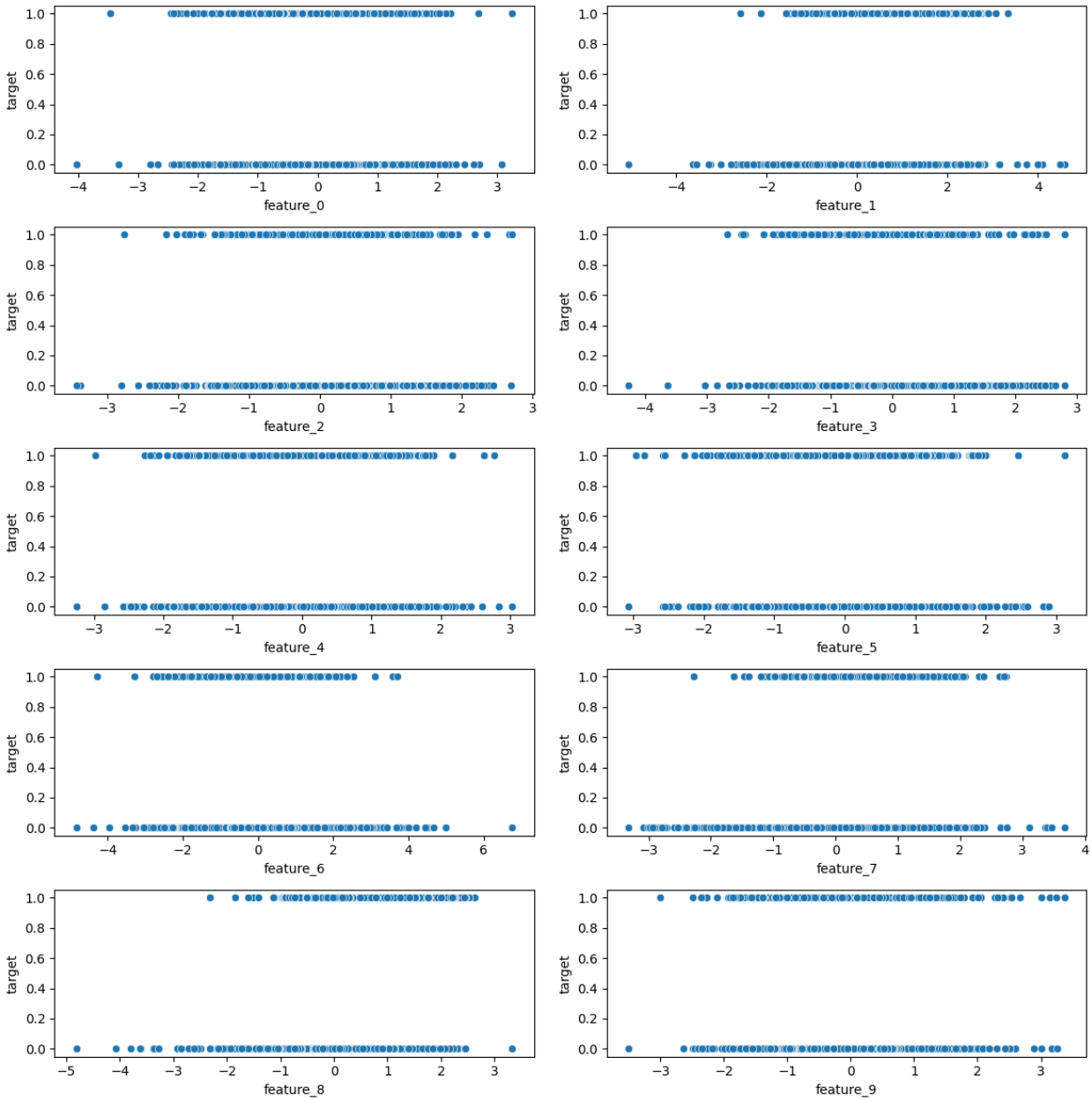

Figure 2: Frecuencia de CV Error

2.1 Modelo Ambas Clases

Ahora bien para el modelo de ambas clases hicimos un 10 fold cross validation usando la función *StratifiedK-Fold* ya que el dataframe no se encuentra balanceado. Con lo anterior, calculamos las métricas de: accuracy, precision, recall, f1 y specificity para analizar el desempeño de tres modelos distintos: regresión logísitica, k nearest neighbors y random forest que eran aquellos que mejor se desempeñan para una categorización binaria. A continuación adjuntamos las métricas obtenidas con cada uno de los algoritmos.

Resultados de los modelos					
Modelo	Accuracy	Precision	Recall	F1	Specificity
Logistic Regression	0.751	0.6895	0.3256	0.4372	0.9317
Random Forest	0.8100	0.7803	0.5036	0.6108	0.9401
5 nearest neighbor	0.748	0.8335	0.837	0.8296	-

Table 4: Comparación de los resultados de los modelos

Antes de hacer el análisis de las métricas recordemos que significan cada una de estas:

1. **Accuracy (Exactitud)**: Mide la proporción de predicciones correctas realizadas por el modelo sobre el total de predicciones. Es una medida general de la precisión del modelo.

2. **Precision (Precisión)**: Indica la proporción de predicciones positivas correctas (verdaderos positivos) sobre el total de predicciones positivas realizadas por el modelo. Es útil cuando el costo de los falsos positivos es alto.

3. **Recall (Recuperación o Sensibilidad)**: Representa la proporción de casos positivos reales (verdaderos positivos) que fueron correctamente identificados por el modelo sobre el total de casos positivos reales. Es útil cuando el costo de los falsos negativos es alto.

4. **F1 Score**: Es la media harmónica de la precisión y el recall. Proporciona un equilibrio entre la precisión y el recall, lo que lo hace útil cuando hay un desequilibrio en la distribución de clases en los datos.

5. **Specificity (Especificidad)**: Indica la proporción de casos negativos reales (verdaderos negativos) que fueron correctamente identificados por el modelo sobre el total de casos negativos reales. Es complementaria al recall y es útil cuando el foco está en la capacidad del modelo para predecir correctamente los casos negativos.

Por lo tanto la medida más importante a tener en cuenta para optimizar el modelo para ambas clases es el accuracy y este es mayor en el algoritmo del random forest propuesto.

2.2 Modelo Clase 0

Ahora bien, para calcular la clase 0 solo tuvimos en cuenta el modelo de regresión logísitica y el de random forest ya que el modelo de kNN no permite cambiar los pesos de las clases dentro de los parámetros y la verdad el cambio de estos en el sistema de votación sería algo arbitrario y un mal indicador. Asimismo, estos se añadieron como un parámetro adicional a los modelos de la siguiente forma:

```
1
2 logreg = LogisticRegression(class_weight={0: 10, 1: 1})
3
4 rf = RandomForestClassifier(n_estimators=100, random_state=42, class_weight={0: 10, 1: 1})
```

Para lo cual, los resultados obtenidos para los dos modelos usados fueron igual a:

Resultados de los modelos					
Modelo	Accuracy	Precision	Recall	F1	Specificity
Regresión Logística	0.701	0.1	0.0033	0.0065	0.0033
Random Forest	0.801	0.7250	0.5439	0.6169	0.9103

Table 5: Comparación de los resultados de los modelos

En este caso el random forest también presenta unos indicadores superiores pero específicamente en specificity que es donde más nos importa para los casos negativos, es decir la clase 0.

2.3 Modelo Clase 1

De forma análoga, solo tuvimos en cuenta el modelo de regresión logísitica y el de random forest ya que el modelo de kNN no permite cambiar los pesos de las clases dentro de los parámetros y la verdad el cambio de estos en el sistema de votación sería algo arbitrario y un mal indicador. Asimismo, estos se añadieron como un parámetro adicional a los modelos de la siguiente forma pero dándole el valor de 10 al 1:

```
1
2 logreg = LogisticRegression(class_weight={0: 1, 1: 10})
3
4 rf = RandomForestClassifier(n_estimators=100, random_state=42, class_weight={0: 1, 1: 10})
```

Para lo cual, los resultados obtenidos para los dos modelos usados fueron igual a:

Resultados de los modelos					
Modelo	Accuracy	Precision	Recall	F1	Specificity
Regresión Logística	0.398	0.3277	0.9697	0.4898	0.9697
Random Forest	0.784	0.7221	0.4563	0.5538	0.9231

Table 6: Comparación de los resultados de los modelos

Finalmente, n este caso el random forest también presenta unos indicadores superiores pero específicamente en precision y recall que es donde más nos importa para los casos positivos, es decir la clase 1.