Trabajo Práctico - Segunda entrega
Organización de Datos 75.06/95.58
Cátedra Collinet

Alumnos: Rodriguez, Alejo y Vergara, Ariel

Padrones: 99869 y 97010

Corrector: Collinet, Jorge

Preprocesamientos

En este trabajo práctico los pre procesamientos no se realizaron mediante una función, sino que para cada uno de ellos se creó una clase que hereda de BaseEstimator y TransformerMixin (ambas clases de sklearn). Se hizo esto para poder utilizar el preprocesamiento como el primer paso de un Pipeline de sklearn, de manera tal que el siguiente y último paso sea el modelo. De esta manera, al hacer K Fold Cross Validation y Grid Search CV, en cada paso se puede fitear tanto el preprocesamiento (en caso de que utilice algún valor para rellenar datos o un LabelEncoder por ejemplo) como el modelo con los datos de entrenamiento, para luego realizar la predicción sobre el set de validación correspondiente. A continuación se encuentra la tabla con los pre procesamientos utilizados:

Preprocesamientos		
Nombre	**Descripción**	**Clase**
PreprocessingLE	● Agrega una columna fila_isna (que indica si el valor de la fila es o no NaN). ● Elimina columnas sin información valiosa[1]. ● Agrega una columna nombre_sede_isna (que indica si el valor de la fila es o no NaN) y rellena los NaN con la moda. ● Encodea variables categóricas mediante LabelEncoding[2]. ● Agrega una columna edad_isna (que indica si el valor de la fila es o no NaN). ● Completa los missing values de la columna edad con la media. ● Convierte en bins los valores de las columnas edad y precio_ticket.	PreprocessingLE
PreprocessingOHE	● Agrega una columna fila_isna que indica si el valor de la fila es o no NaN. ● Elimina columnas sin información valiosa[1]. ● Encodea variables categóricas mediante OneHotEncoding[2]. ● Agrega una columna edad_isna (que indica si el valor de la fila es o no NaN). ● Completa los missing values de la columna edad con la media. ● Convierte en bins los valores de la columna precio_ticket.	PreprocessingOHE
PreprocessingGaussianNB1	● Elimina columnas sin información valiosa[1] y con valores categóricos[2]. ● Se agrega la columna acompaniantes y se eliminan las columnas parientes y amigos. ● Completa los missing values de la columna edad con la media.	PreprocessingGaussianNB1
PreprocessingCategoricalNB1	● Elimina columnas sin información valiosa[1] y con valores continuos o discretos[4]. ● Encodea variables categóricas mediante LabelEncoding[2].	PreprocessingCategoricalNB1

PreprocessingSE	• Agrega una columna fila_isna que indica si el valor de la fila es o no NaN. • Elimina columnas sin información valiosa[1]. • Encodea variables categóricas mediante OneHotEncoding[2]. • Completa los missing values de la columna edad con la media. • Escala los valores numéricos[3] a media 0 y desvío estándar 1 con StandardScaler.	PreprocessingSE
PreprocessingSE_2	• Agrega una columna fila_isna que indica si el valor de la fila es o no NaN. • Elimina columnas sin información valiosa[1]. • Encodea variables categóricas mediante LabelEncoding[2]. • Completa los missing values de la columna edad con la media. • Escala los valores numéricos[3] a media 0 y desvío estándar 1 con StandardScaler.	PreprocesingSE_2
PreprocessingXGBoost	• Agrega una columna fila_isna que indica si el valor de la fila es o no NaN. • Agrega una columna edad_isna que indica si el valor de la edad es o no NaN. • Elimina columnas sin información valiosa[1]. • Encodea variables categóricas mediante OneHotEncoding[2]. • Completa los missing values de la columna edad con la media. • Convierte en bins los valores de las columnas edad y precio_ticket.	PreprocesingXGBoost
PreprocessingXGBoost2	• Agrega una columna fila_isna que indica si el valor de la fila es o no NaN. • Agrega una columna edad_isna que indica si el valor de la edad es o no NaN. • Elimina columnas sin información valiosa[1]. • Encodea variables categóricas mediante OneHotEncoding[2]. • No completa missing values.	PreprocessingXGBoost2
PreprocessingCategoricalNB2	• Elimina columnas sin información valiosa[1] y con valores continuos o discretos[4]. • Encodea variables categóricas mediante LabelEncoding[2]. • Transforma en bins los valores de las columnas edad y precio_ticket.	PreprocessingCategoricalNB2
PreprocessingCategoricalNB3	• Elimina columnas sin información valiosa[1] y con valores continuos o discretos[4]. • Encodea variables categóricas mediante OneHotEncoding.	PreprocessingCategoricalNB3

Aclaraciones:

- [1] fila, id_usuario y id_ticket
- [2] genero, nombre_sala, tipo_de_sala
- [3] edad, precio_ticket, parientes y amigos
- [3] edad, precio_ticket, parientes y amigos

Los preprocesamientos PreprocessingSE_2, PreprocessingXGBoost, PreprocessingXGBoost2, PreprocessingCategoricalNB2 y PreprocessingCategoricalNB3 fueron descartados luego de comparar las métricas obtenidas mediante Cross Validation con las de otros preprocesamientos, siempre utilizando los mismos modelos para cada uno.

Queda pendiente un refactor para que las diferentes transformaciones que se hacen en cada una de las clases de la tabla sean realizadas por diferentes Transformers. De esta forma, en lugar de que cada preprocesamiento sea una clase, se podría hacer que sea una función que devuelve un Pipeline de preprocesamiento con todos los steps correspondientes al mismo (ej: transformar la edad, eliminar las columnas que no aportan información, etc)

Modelos

Métricas

Para comparar los modelos entre sí se realizó un K Fold Cross Validation de 8 folds sobre todos los datos de entrenamiento. Una vez seleccionado el modelo, se dividieron los datos en train y test (con un test size de 15% debido a los pocos datos con los que se cuenta), se entrenó el modelo correspondiente con el train y se realizó la predicción sobre el test. Para todos los modelos, se utilizó el mismo train_test_split y para todos los Cross Validation, los mismos folds.

A partir de la predicción sobre el split de test, se obtuvieron las métricas presentadas en la siguiente tabla:

Modelos						
Nombre	**Preprocesamiento**	**Métricas**				
		AUC-ROC	**Accuracy**	**Precisión**	**Recall**	**F1 Score**
0-Baseline	-	0.8446	0.8512	0.9375	0.6521	0.7692
1-ArbolDeDecision	PreprocessingLE	0.9084	0.8595	0.8536	0.7608	0.8045
2-RandomForest	PreprocessingLE	0.9089	0.8760	0.9428	0.7173	0.8148
3-NaiveBayes	PreprocessingGaussianNB1 - PreprocessingCategoricalNB1	0.8759	0.8181	0.8333	0.6521	0.7317
4-XGBoost	PreprocessinLE	0.8971	0.8677	0.8750	0.7608	0.8139
5-SVM	PreprocessingSE	0.8840	0.8429	0.8139	0.7607	0.7865
6-KNN	PreprocessingSE	0.8708	0.8347	0.8611	0.6739	0.7560
7-Perceptron	PreprocessingSE	0.6071	0.4793	0.3975	0.7173	0.5116

8-RedNeuronal	PreprocessingSE	0.8718	0.8347	0.8611	0.6739	0.7560
9-Ensambles	PreprocessingLE - PreprocessingSE	0.9127	0.8595	0.8372	0.7826	0.8089

Decisiones de diseño

Para los modelos basados en árboles (1-ArbolDeDecision, 2-RandomForest, 4-XGBoost) se decidió utilizar el preprocesamiento basado en OneHotEncoding para los features categóricos, anteriormente se habían implementado con LabelEncoding pero se descartó debido a que el modelo puede inferir erróneamente una relación de orden entre los mismos.

Tanto para KNN como para SVM y Perceptron se utilizaron preprocesamientos que estandarizan los features numéricos ya que tienen un mejor rendimiento de esa manera, evitando que el modelo considere más o menos importantes a los features dependiendo de su escala.

Para la RedNeuronal se utilizó el mismo preprocesamiento con StandardScaler. A diferencia de lo realizado con otros modelos, en este no se utilizaron algoritmos de búsqueda de hiperparametros debido al tiempo que consumían los mismos. De todas formas se verificaron distintas configuraciones de capas, neuronas, optimizadores y regularizadores para llegar al mejor resultado, obteniendo las métricas de cada uno mediante Cross Validation.

En NaiveBayes se indican dos pre procesamientos debido a que el modelo presentado consiste en un ensamble Stacking con dos Naive Bayes, un CategoricalNB y un GaussianNB. El estimador final del ensamble corresponde a otro GaussianNB, el cual utiliza las salidas de los dos modelos anteriores para obtener un mejor resultado que cada uno individualmente. En cuanto al procesamiento utilizado en CategoricalNB, este solo deja variables categóricas encodeadas mediante OneHotEncoding o binarias (como por ejemplo fila_isna). Por el contrario, el preprocesamiento utilizado con GaussianNB solamente preserva variables discretas o continuas.

Finalmente, el modelo correspondiente a Ensambles es, al igual que al anterior, un ensamble de tipo Stacking que utiliza como estimador final un GaussianNB. Los modelos utilizados en la primera capa son los correspondientes a los mejores resultados de SVM, XGBoost y RandomForest.

Análisis de los resultados

Salvo el caso del Perceptron, para el cual los resultados obtenidos en las métricas fueron muy malos (debido a que es altamente probable que los datos no sean linealmente separables), los resultados de los modelos fueron muy similares en cuanto al valor del AUC-ROC. Estos rondan entre 0.87 y 0.91.

Los valores más altos fueron obtenidos por el Árbol de Decisión, Random Forest, XGBoost y el Ensamble. En el caso de los últimos tres modelos mencionados, el hecho de que todos sean un ensamble ayuda a obtener mejores resultados, dado que se combinan los resultados de varios clasificadores. Sin embargo, en el caso del Árbol de Decisión, no se encontró una explicación de por qué obtuvo tan buenos resultados, siendo que es un modelo básico y sencillo y se suele utilizar como una solución inicial para un problema. Obtuvo incluso un resultado similar en cuanto a su AUC-ROC al del Random Forest, que es justamente un ensamble de Árboles de Decisión. Se cree que puede haber sobre ajustado mucho al dataset de entrenamiento y que la partición utilizada haya ayudado a que obtenga

un muy buen resultado.

Cambio de Label Encoder a One Hot Encoder

Inicialmente, en los modelos basados en árboles (Árbol de Decisión, Random Forest y XGBoost) y en NB se utilizó Label Encoder en lugar de One Hot Encoder para encodear las variables categóricas. Esto se debió a que, al probar ambos tipos de encoding, el LE obtuvo mejores resultados para nuestro set de datos que OHE. Sin embargo, encodear las variables categóricas no binarias presentes en nuestro set de datos (que corresponden al nombre de la sede y al tipo de sala) utilizando LE no es teóricamente correcto, dado que se introduce una relación de orden en la variable donde previamente no la había (por ejemplo, fiumark_palermo no es mayor a fiumar_chacarita). El hecho de haber obtenido mejores resultados en estos modelos con las implementaciones que utilizan el preprocesamiento PreprocessingLE puede haberse debido a la conformación de nuestro set de datos, por lo que no es conveniente utilizarlo.

A continuación se muestran los resultados obtenidos para estos modelos, una vez reemplazado el preprocesamiento para que utilice OHE:

Modelos						
Nombre	**Preprocesamiento**	**Métricas**				
		AUC-ROC	**Accuracy**	**Precisión**	**Recall**	**F1 Score**
1-ArbolDeDecision	PreprocessingOHE	0.8756	0.8512	0.8333	0.7608	0.7954
2-RandomForest	PreprocessingOHE	0.8962	0.8677	0.9166	0.7173	0.8048
3-NaiveBayes	PreprocessingGaussianNB1 - PreprocessingCategoricalNB3	0.8808	0.8099	0.8709	0.5869	0.7012
4-XGBoost	PreprocessinOHE	0.8921	0.8429	0.8857	0.6739	0.7654
9-Ensambles	PreprocessingOHE - PreprocessingSE	0.9072	0.8595	0.8372	0.7826	0.8089

Salvo el caso del Árbol de Decisión, en el cual su AUC-ROC disminuyó un 0.03, los demás modelos obtuvieron resultados muy similares a cuando utilizaban LE. A su vez, el Ensamble obtuvo los mismos valores en todas las métricas salvo en AUC-ROC donde disminuyó levemente, por lo que no influenció en el resultado final modificar el tipo de encoding utilizado tanto en XGBoost y en Random Forest. Por el contrario, Naive Bayes obtuvo un Precisión más alto pero un peor Recall (aumentó la tasa de TN pero también la de FN).

Con estos resultados, el modelo con mayor AUC-ROC sigue siendo el Ensamble.

Conclusión

Para la entrega final, se elige el modelo correspondiente a 9-Ensambles, el cual obtuvo el mejor AUC-ROC entre todos los modelos entrenados. Este modelo no solo es un ensamble, sino que dos de los modelos que utiliza para realizarlo también lo son, siendo estos XGBoost y Random Forest. Al ser un ensamble, cada uno de los modelos que lo conforma sobre ajusta a su manera a los datos de entrenamiento, lo cual permite generar un mejor clasificador al combinarlos.

Si se desea también un buen resultado en Presition, métrica que nos permite asegurarnos que tan probable es que un caso marcado como positivo (volvería a ver Frozen 4) sea efectivamente positivo, se recomienda el uso del modelo 2-RandomForest. El valor de AUC-ROC de este modelo es bastante similar al obtenido por el 9-Ensambles, pero su valor de Presition es considerablemente más alto. Sin embargo, el valor obtenido en Recall es menor, por lo cual aumenta la probabilidad de obtener falsos negativos.

En comparación al baseline de la primera entrega, el modelo final obtenido es más complejo y puede inferir relaciones a partir de los datos que no podemos detectar a simple vista (como se hizo en la primera parte del Trabajo Práctico). El baseline, además de ser más simple, se construyó a partir de todos los datos provistos y se evaluó sobre los mismos, por lo que existe una gran probabilidad de overfitting. Por el contrario, el ensamble elegido se entrenó sobre una parte de nuestros datos elegida al azar y se validó contra la parte restante.