Práctica 3

Inteligencia de Negocio

Amin Kasrou Aouam

UNIVERSIDAD
DE GRANADA

2021-01-12

Índice

Práctica 3

Introducción

En esta práctica, resolveremos un problema de clasificación multiclase, en concreto, trataremos de predecir la categoría de precio de una serie de coches

Preprocesamiento de datos

Valores nulos

Nuestro *dataset* contiene bastantes valores nulos, optamos por estrategias diferentes según las columnas:

- Eliminación: tipo marchas, descuento, ciudad
- Imputación: asientos, motor cc, potencia

El criterio que seleccionamos es el número de instancias nulas, en el caso de que sean muchas optamos por imputar, para mantener un número adecuado de datos.

La implementación se encuentra en la siguiente función:

```
def process_null_values(df_list):
    drop_columns = ["tipo_marchas", "descuento", "ciudad"]
    fill_columns = ["asientos", "motor_cc", "potencia"]
    for df in df_list:
        for column in fill_columns:
            if column == "asientos":
                df[column].fillna(value=df[column].median(), inplace=
                    True)
            else:
                df[column].fillna(
                    value=df[column].str.extract("(\d+)").mean(),
                        inplace=True
                )
        df.drop(columns=drop_columns, inplace=True)
        df.dropna(inplace=True)
    return df_list
```

Valores no numéricos

Ciertas columnas contienen valores alfanúmericos, aunque se nos proporcionan distintos archivos CSV para realizar un *mapping*. En este caso, utilizamos un **LabelEncoder**, y como entrada le damos el CSV correspondiente.

Es primordial usar el mismo *LabelEncoder* para los datos de entrenamiento como de test. La implementación se encuentra en la siguiente función:

```
def encode_columns(df_list):
    label_encoder = LabelEncoder()
    files = [
        "ao",
        "asientos",
        "combustible",
        "consumo",
        "kilometros",
        "mano",
        "motor_cc",
        "nombre",
        "potencia",
    ]
    for data in files:
        for df in df_list:
            label = label_encoder.fit(read_csv("data/" + data + ".csv",
                squeeze=True))
            if data == "ao":
                df["año"] = label.transform(df["año"])
            else:
                df[data] = label.transform(df[data])
    return df_list
```

Balanceo de clases

Observamos que la mayoría de coches son de la categoría de precio 3, lo cual no es idóneo para entrenar un modelo de inteligencia artificial.

Debemos realizar un balanceo de las clases, en este caso optamos por usar el modelo **SMOTEEEN**, que combina un *over-sampling* mediante **SMOTE** y una limpieza gracias a *Edited Nearest Neighbours (ENN)*.

La implementación se encuentra en esta función:

```
def balance_training_data(df):
    smote_enn = SMOTEENN(random_state=42)
    data, target = split_data_target(df=df, dataset="data")
    balanced_data, balanced_target = smote_enn.fit_resample(data,
       target)
    balanced_data_df = DataFrame(
        balanced_data, columns=df.columns.difference(["precio_cat"])
    )
    balanced_target_df = DataFrame(balanced_target, columns=["
       precio_cat"])
    return balanced_data_df, balanced_target_df
```

Elección de algoritmo

Elegimos el algoritmo **GradientBoostingClassifier**, que pertenece a los algoritmos de *ensemble*. Éstos combinan las predicciones de varios clasificadores, con el objetivo de mejorar la generalización y la robustez de las predicciones.

En particular, pertenece a la familia de *boosting methods*, cuya característica es que los clasificadores se crean de forma secuencial, y uno de ellos trata de reducir el sesgo de los demás.

Resultados obtenidos

Al ejecutar el programa en local obtenemos los siguientes resultados:

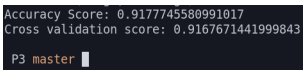

Figura 1: Resultados de ejecución

Desafortunadamente, en la plataforma Kaggle obtenemos unos resultados pésimos:

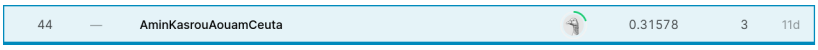

Figura 2: Resultados de Kaggle

Análisis de resultados

Debido a la discrepancia entre los resultados de la ejecución en local, y de la plataforma Kaggle, intuimos que debe de haber un problema en el preprocesamiento de datos.

También es posible que el modelo no sea óptimo para la tarea, aunque no justificaría un rendimiento tan bajo, puede contribuir a ello.