ITBA
Instituto Tecnológicc
de Buenos Aires

airbnb

82.05 - Análisis Predictivo

12/06/2023

COMPETENCIA

PREDICCIÓN DE SCORE DE ALOJAMIENTO

MANUEL SAÚL HANONO

Score obtenido:

0.72347	0.71948

1	▲ 8	MANUEL SAUL HANONO		0.72347

01 Introducción

Composición y preparación de la base, desafíos y objetivos del trabajo.

02 Tratamiento del dataset

EDA, tratamiento de columnas, imputaciones y codificación de variables.

03 Selección de modelos

Modelos utilizados, comparaciones y proceso de selección.

04 Conclusiones

Insights y análisis del modelo ganador.

INTRODUCCIÓN

La base contiene información sobre alojamientos publicados en Airbnb en la ciudad de **Amsterdam**.

En total, la base de *train* cuenta con 4.928 registros con, en un principio, 67 variables.

La base de *val* cuenta con 1.233 registros, con 66 variables, que se utilizarán para predecir la variable ***review_scores_rating***.

PREPARACIÓN DE LA BASE

- Se realiza un análisis exploratorio completo de la base.
- Se ha verificado que la base no contiene duplicados.
- Se han ~~eliminado~~ 3 columnas cuyos valores eran todos nulos.
- Se realizan *transformaciones particulares* para varias de las columnas.
- Se toma una decisión con respecto a los **datos faltantes** de cada columna.
- Se unifica un criterio para convertir *variables categoricas a númericas*.

TRATAMIENTO DE COLUMNAS DE TEXTO

Para el caso del **nombre** (*name*), la **descripción** (*description*) y el **detalle del barrio** (*neighborhood_overview*) de la **publicación** y para la **descripción del host** (*host_about*), se generan dos columnas por cada una que permitan generar valor y hagan que sea posible incluirlas en el modelo.

```
#Impacto de la longitud:
df['name_length'] = df['name'].apply(len)

#Análisis de sentimiento:
names = df['name']
clean_names = names.str.replace('[^\w\s]','').str.lower().str.split()
sentiments = []
for name in clean_names:
    sentiment = 0
    for word in name:
        sentiment += TextBlob(word).sentiment.polarity
    sentiments.append(sentiment)
#Agregar los puntajes de sentimiento como una nueva columna al dataframe
df['name_sentiment'] = sentiments
```

TRATAMIENTO DE COLUMNAS DE TEXTO II

Para los otros caso de las columnas relacionadas con el host se realizaron tratamientos particulares, tales como los siguientes para el **nombre** (*host_name*), **ubicación** (*host_location*) y **barrio** (*host_neighbourhood*).

```python
#Categorizar los nombres según la letra en la que comienzan:
def get_first_letter(name):
    return name[0].upper()
df['host_name_first_letter'] = df['host_name'].apply(get_first_letter)
df.host_name_first_letter.value_counts()

#Utilizo la libreria gender_guesser para reconocer si el host es hombre o mujer.
import gender_guesser.detector as gender
d = gender.Detector()
#Función para predecir género
def predict_gender(name):
    gender = d.get_gender(name)
    if gender == 'male':
        return 'Male'
    elif gender == 'female':
        return 'Female'
    if pd.isna(name):
        return 'Missing'
    else:
        return 'Unknown'
# Aplicar la función a la columna 'host_name'
df['predicted_gender'] = df['host_name'].apply(predict_gender)
```

```python
#Función para clasificar los valores
def classify_location(location):
    if pd.isna(location):
        return 'Missing'
    elif 'Amsterdam' in str(location):
        return 'Amsterdam'
    elif 'Netherlands' in str(location):
        return 'Netherlands'
    else:
        return 'Other'

#Aplicar la función a la columna 'host_location'
df['host_location'] = df['host_location'].apply(classify_location)
```

```python
df.host_neighbourhood.value_counts()
df['same_neighbourhood'] = df.apply(lambda row: 't' if row['host_neighbourhood'] ==
                                    row['neighbourhood_cleansed'] else 'f', axis=1)

df.same_neighbourhood.value_counts()
```

TRATAMIENTO DE COLUMNAS III

Finalmente, se resolvieron particularidades de columnas con **fechas**, **precios** y **porcentajes**, se realizó un tratamiento especial para la variable de **baños** (*bathrooms_text*) y un conteo para los **amenities**.

```
import re
df['bathrooms_number'] = df['bathrooms_text'].str.extract(r'(\d+\.?\d*)').astype(float)
df['bathrooms_number'] = df['bathrooms_number'].fillna(0.5) #Ya que los que no tenian numero, decian half_bath.

#Función para clasificar los valores
def classify_bath(bath):
    if 'shared' in str(bath).lower():
        return 'shared'
    else:
        return 'private'
#Aplicar la función a la columna 'host_location'
df['bathrooms_type'] = df['bathrooms_text'].apply(classify_bath)
df.loc[df['bathrooms_number'] == 0, 'bathrooms_type'] = 'none'
```

```
#amenities: Las amenities de la propiedad. Decido contarlas.
df.amenities.value_counts()
df['amenities_count'] = df['amenities'].str.count(',') + 1
```

DATOS FALTANTES Y VARIABLES CATEGORICAS

Como parte final de la preparación, se genero un **Pipeline** para **imputar missings** y realizar el **encoding** de las variables categoricas.

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.compose import make_column_transformer

categorical_columns = [c for c in df.columns if df[c].dtype == 'object']
numerical_columns = [c for c in df.columns if c not in categorical_columns and c != 'review_scores_rating' and c != 'i

categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='constant', fill_value='Missing')),
    ('encoder', OrdinalEncoder())
])

numerical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent'))
])

df2 = df

df2[categorical_columns] = categorical_pipeline.fit_transform(df2[categorical_columns])
df2[numerical_columns] = numerical_pipeline.fit_transform(df2[numerical_columns])

df2.head()
```

ENTRENAMIENTO DE LOS MODELOS

Tras realizar todas estas modificaciones, se obtiene un dataset de train con 4968 registros y 65 variables. Como baseline para todos los modelos que fueron probados, se realizó una partición de **80-20** del dataset de train.

```
X = train.drop('review_scores_rating', axis=1)
y = train['review_scores_rating']

#Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

PROCESO DE SELECCIÓN DE MODELOS

- Regresión Lineal Simple
- Regresión Lineal Múltiple
- XGBoost
- LightGBM
- Random Forest Regressor
- Extra Tree Regressor
- **SVR (Support Vector Regression)**

MODELO GANADOR - SVR

```python
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from skopt import BayesSearchCV
from sklearn.preprocessing import MinMaxScaler

X = train.drop('review_scores_rating', axis=1)
y = train['review_scores_rating']

#Normalizar los datos de entrada
scaler = MinMaxScaler()
X_normalized = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_normalized, y, test_size=0.2)

param_space = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'epsilon': (0.01, 1.0),
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': (0.001, 10.0),
    'degree': (1, 5),
}
model = SVR()

#Realizar la búsqueda bayesiana de hiperparámetros y ajustar el mejor modelo automáticamente
opt = BayesSearchCV(model, param_space, n_iter=50, cv=5, scoring='neg_mean_squared_error', refit=True)
opt.fit(X_train, y_train)

#Obtener el mejor modelo y los mejores parámetros
best_model = opt.best_estimator_
best_params = opt.best_params_

#Realizar predicciones en el conjunto de prueba
y_pred = best_model.predict(X_test)

# Calcular el error cuadrático medio (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error: {:.4f}".format(mse))

# Calcular el coeficiente de determinación (R^2)
r2 = r2_score(y_test, y_pred)
print("Coefficient of Determination (R^2): {:.2f}".format(r2))
```

```
Best Parameters:
C: 0.1
degree: 4
epsilon: 0.11537046867819958
gamma: 2.511158244626176
kernel: linear
```

ITBA
Instituto Tecnológico de Buenos Aires

airbnb

MUCHAS GRACIAS