Examen: Statistical Learning

Andrés Proaño

19 de octubre de 2025

1. Ejercicio 1: Boston Housing Datasets

Se trabajará con el dataset Boston Housing que contiene información sobre viviendas en Boston, Massachusetts. El objetivo es predecir el valor mediano de las viviendas.

El dataset contiene las siguientes variables:

- CRIM per capita crime rate by town
- ZN proportion of residential land zoned for lots over 25,000 sq. ft.
- INDUS proportion of non-retail business acres per town
- CHAS Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- AGE proportion of owner-occupied units built prior to 1940
- DIS weighted distances to five Boston employment centres
- RAD index of accessibility to radial highways
- TAX full-value property-tax rate per $10,000
- PTRATIO pupil-teacher ratio by town
- B $1000(Bk - 0{,}63)^2$ where Bk is the proportion of blacks by town
- LSTAT porcentaje de población de bajo estatus
- MEDV Median value of owner-occupied homes in $1000's

1.1. Base de Datos

Se utilizó el conjunto de datos Boston Housing disponible en la Universidad de Toronto con fines educativos.

```
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=
    None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2,
    :2]])
target = raw_df.values[1::2, 2]
```

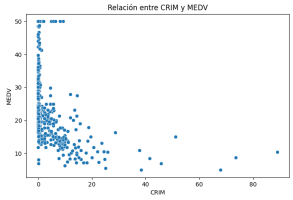
1.2. Objetivo

El objetivo es predecir el valor mediano de las viviendas comparando diferentes modelos y sus rendimientos.

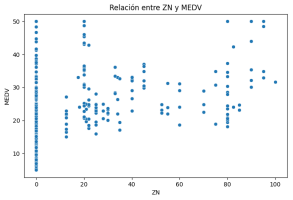
1.3. Tareas Requeridas

1.3.1. Implementación de Modelos

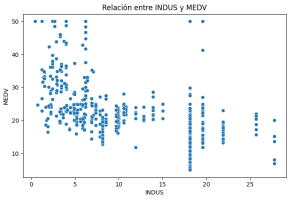
Antes de realizar los diferentes modelos, vamos a buscar significancia dentro de las variables para evitar utilizar datos que no aporten al trabajo.



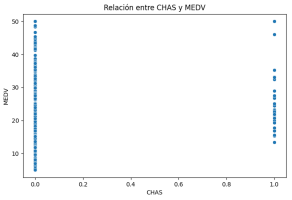
(a) Tasa de criminalidad per cápita



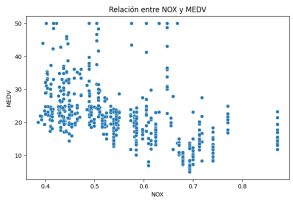
(b) Proporción de terrenos residenciales



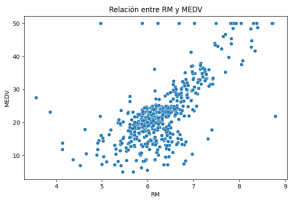
(c) Proporción de terrenos industriales



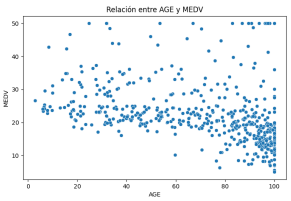
(d) Variable Dummy del Río Charles



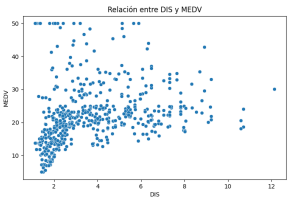
(e) Concentración de Óxidos de Nitrógeno



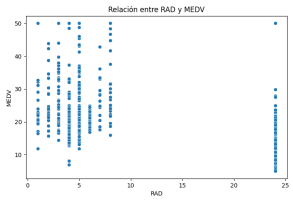
(f) Número promedio de habitaciones



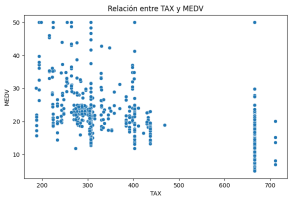
(g) Edad



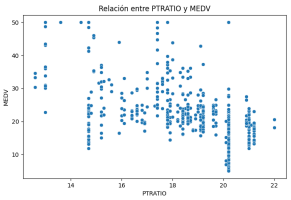
(h) Distancia a centros de empleo



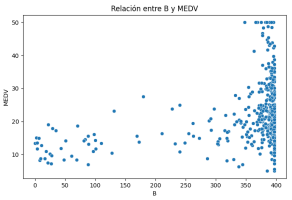
(i) Accesibilidad a autopistas radiales



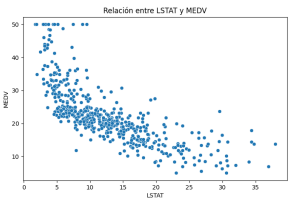
(j) Tasa de impuesto predial



(k) Relación de Población por Área



(l) Proporción de población negra



(m) Porcentaje de población de bajo estatus

Figura 1: Distribución de variables por Valor Mediano de Viviendas

Como podemos observar, todas las variables son significativas al momento de tomar en cuenta el precio de la casa. Por lo tanto, se utilizarán todas las variables para los diferentes modelos.

1.4. Modelo lineal Gaussiano (MLG)

Para realizar el MLG utilizamos el siguiente código:

```
splitter = ShuffleSplit(n_splits=50, test_size=0.3, random_state
    =42)

mae_scores_mlg, rmse_scores_mlg = [], []

for train_index, test_index in splitter.split(x):
    X_train, X_test = x[train_index], x[test_index]
    y_train, y_test = y[train_index], y[test_index]

    X_train_sm = sm.add_constant(X_train, has_constant='add')
    X_test_sm = sm.add_constant(X_test, has_constant='add')

    model_gaussian = sm.GLM(y_train, X_train_sm, family=sm.
        families.Gaussian())
    fitted = model_gaussian.fit()

    y_pred_gaussian = fitted.predict(X_test_sm)

    mae_scores_mlg.append(mean_absolute_error(y_test,
        y_pred_gaussian))
    rmse_scores_mlg.append(mean_squared_error(y_test,
        y_pred_gaussian) ** 0.5)


mae_median = np.median(mae_scores_mlg)
mae_std = np.std(mae_scores_mlg, ddof=1)
rmse_median = np.median(rmse_scores_mlg)
rmse_std = np.std(rmse_scores_mlg, ddof=1)

print(f"Gaussian Model MAE: {mae_median:.2f} +/- {mae_std:.2f}")
print(f"Gaussian Model RMSE: {rmse_median:.2f} +/- {rmse_std:.2f}
    ")
```

Obteniendo los siguientes resultados:

Cuadro 1: Resultados del Modelo Lineal Gaussiano

Métrica	**Valor**
MAE	3.39 +- 0.22
RMSE	4.91 ± 0.45

1.4.1. Interpretación de los resultados

Los resultados del modelo lineal Gaussiano indica que la diferencia entre los valores reales de MEDV y los valores estimados por el modelo es de 3.39 mil dólares, con una variabilidad moderadamente baja. El RMSE, al ser más sensible a errores grandes, indica

que las desviaciones cuadráticas promedio equivalen a unos 4.91 mil dólares, lo que sugiere que existen algunos errores más grandes, aunque no dominantes.

1.5. Modelo lineal con regularización Elastic Net

Para la realización utilizamos el siguiente código:

```
param_grid = {
    "elastic__alpha": [0.001, 0.01, 0.1, 1, 10],
    "elastic__l1_ratio": [0.1, 0.3, 0.5, 0.7, 0.9, 0.99],
}

pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("elastic", ElasticNet(max_iter=5000, random_state=42))
])

inner_cv = KFold(n_splits=7, shuffle=True, random_state=42)
splitter = ShuffleSplit(n_splits=50, test_size=0.3, random_state
   =42)

mae_scores, rmse_scores = [], []

for train_idx, test_idx in splitter.split(X_sm.values):
    X_train, X_test = X_sm.iloc[train_idx], X_sm.iloc[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    grid = GridSearchCV(
        pipeline,
        param_grid,
        scoring="neg_mean_squared_error",
        cv=inner_cv,
        n_jobs=-1
    )
    grid.fit(X_train, y_train)

    y_pred = grid.predict(X_test)

    mae_scores.append(mean_absolute_error(y_test, y_pred))
    rmse_scores.append(mean_squared_error(y_test, y_pred) ** 0.5)

mae_median = np.median(mae_scores)
mae_std = np.std(mae_scores, ddof=1)
rmse_median = np.median(rmse_scores)
rmse_std = np.std(rmse_scores, ddof=1)

print(f"Elastic Net MAE: {mae_median:.2f} +/- {mae_std:.2f}")
print(f"Elastic Net RMSE: {rmse_median:.2f} +/- {rmse_std:.2f}")
```

Obteniendo los siguientes resultados:

Cuadro 2: Métricas de Rendimiento del Elastic Net

Métrica	Valor
MAE	3.36 ± 0.22
RMSE	4.90 ± 0.47

1.5.1. Interpretación de Resultados

El modelo de regresión Elastic Net, tras realizar 50 particiones aleatorias de validación cruzada, arroja un error absoluto medio de 3.36 mil dólares y un error cuadrático medio de 4.90 mil dólares sobre la variable objetivo MEDV. La dispersión de las métricas ($\pm$ 0.22 y $\pm$ 0.47 respectivamente) es relativamente baja, lo que indica que el modelo mantiene un rendimiento estable ante diferentes divisiones de entrenamiento y prueba.

1.6. Modelo lineal con componentes principales (PCA)

```
pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("pca", PCA()),
    ("linreg", LinearRegression())
])

param_grid = {
    "pca__n_components": [5, 7, 9, 11, None]
}

inner_cv = KFold(n_splits=7, shuffle=True, random_state=42)
splitter = ShuffleSplit(n_splits=50, test_size=0.3, random_state
   =42)

mae_scores, rmse_scores = [], []

for train_idx, test_idx in splitter.split(X_sm.values):
    X_train, X_test = X_sm.iloc[train_idx], X_sm.iloc[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    grid = GridSearchCV(
        pipeline,
        param_grid,
        scoring="neg_mean_squared_error",
        cv=inner_cv,
        n_jobs=-1
    )
    grid.fit(X_train, y_train)

    y_pred = grid.predict(X_test)

```

```
    mae_scores.append(mean_absolute_error(y_test, y_pred))
    rmse_scores.append(mean_squared_error(y_test, y_pred)**0.5)

mae_median = np.median(mae_scores)
mae_std = np.std(mae_scores, ddof=1)
rmse_median = np.median(rmse_scores)
rmse_std = np.std(rmse_scores, ddof=1)

print(f"PCA + LR MAE: {mae_median:.2f} +/- {mae_std:.2f}")
print(f"PCA + LR RMSE: {rmse_median:.2f} +/- {rmse_std:.2f}")
```

Obteniendo los siguientes resultados:

Cuadro 3: Métricas de Rendimiento del PCA

Métrica	**Valor**
MAE	3.39 ± 0.22
RMSE	4.91 ± 0.47

1.6.1. Interpretación de Resultados

Esto significa que, en promedio, el modelo comete un error absoluto de aproximadamente 3.4 mil dólares en la predicción de MEDV y un error cuadrático medio de unos 4.9 mil dólares. La baja desviación estándar de ambas métricas indica que el comportamiento es estable frente a diferentes particiones de validación cruzada.

1.7. Maquinas de Vector de Soporte para Regresión (SVR)

Para realizar el SVR utilizamos el siguiente código:

```
svr_pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("svr", SVR())
])

param_grid = {
    "svr__kernel": ["linear", "rbf"],
    "svr__C": [0.1, 1, 10],
    "svr__epsilon": [0.01, 0.1, 1],
    "svr__gamma": ["scale", "auto"]  # solo se usa cuando kernel
       ='rbf'
}

inner_cv = KFold(n_splits=7, shuffle=True, random_state=42)
splitter = ShuffleSplit(n_splits=50, test_size=0.3, random_state
   =42)

mae_scores, rmse_scores = [], []

```

```python
for train_idx, test_idx in splitter.split(X_sm.values):
    X_train, X_test = X_sm.iloc[train_idx], X_sm.iloc[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    grid = GridSearchCV(
        svr_pipeline,
        param_grid,
        scoring="neg_mean_squared_error",
        cv=inner_cv,
        n_jobs=-1
    )
    grid.fit(X_train, y_train)

    y_pred = grid.predict(X_test)

    mae_scores.append(mean_absolute_error(y_test, y_pred))
    rmse_scores.append(mean_squared_error(y_test, y_pred) ** 0.5)

mae_median = np.median(mae_scores)
mae_std = np.std(mae_scores, ddof=1)
rmse_median = np.median(rmse_scores)
rmse_std = np.std(rmse_scores, ddof=1)

print(f"SVR MAE:  {mae_median:.2f} +/- {mae_std:.2f}")
print(f"SVR RMSE: {rmse_median:.2f} +/- {rmse_std:.2f}")
```

Obteniendo los siguientes resultados:

Cuadro 4: Métricas de Rendimiento del Modelo SVR

Métrica	**Valor**
MAE	2.35 ± 0.25
RMSE	3.88 ± 0.61

1.7.1. Interpretación de Resultados

El modelo SVR presenta un error medio absoluto de 2.35 mil dólares y un error cuadrático medio de 3.88 mil dólares al predecir el valor medio de viviendas (MEDV).

La desviación estándar moderadamente baja de ambas métricas ($\pm$ 0.25 y $\pm$ 0.61) indica que el desempeño del modelo es consistente y estable a lo largo de las múltiples particiones de validación cruzada.

1.8. Árbol de decisión para regresión

Para realizar el Árbol de Decisión para regresión utilizamos el siguiente código:

```
dtree = DecisionTreeRegressor(random_state=42)

param_grid = {
    "max_depth": [3, 5, 7, 9, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 5]
}

inner_cv = KFold(n_splits=7, shuffle=True, random_state=42)
splitter = ShuffleSplit(n_splits=50, test_size=0.3, random_state
   =42)

mae_scores, rmse_scores = [], []

for train_idx, test_idx in splitter.split(X_sm.values):
    X_train, X_test = X_sm.iloc[train_idx], X_sm.iloc[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    grid = GridSearchCV(
        dtree,
        param_grid,
        scoring="neg_mean_squared_error",
        cv=inner_cv,
        n_jobs=-1
    )
    grid.fit(X_train, y_train)

    y_pred = grid.predict(X_test)

    mae_scores.append(mean_absolute_error(y_test, y_pred))
    rmse_scores.append(mean_squared_error(y_test, y_pred) ** 0.5)

mae_median = np.median(mae_scores)
mae_std = np.std(mae_scores, ddof=1)
rmse_median = np.median(rmse_scores)
rmse_std = np.std(rmse_scores, ddof=1)

print(f"Decision Tree MAE:  {mae_median:.2f} +/- {mae_std:.2f}")
print(f"Decision Tree RMSE: {rmse_median:.2f} +/- {rmse_std:.2f}"
   )
```

Con este código obtenemos los siguientes resultados:

Cuadro 5: Métricas de Rendimiento del Árbol de Decisión para Regresión

Métrica	**Valor**
MAE	2.97 ± 0.42
RMSE	4.53 ± 0.94

1.8.1. Interpretación de Resultados

El modelo de árbol de decisión presenta un error absoluto medio de 2.97 mil dólares y un error cuadrático medio de 4.53 mil dólares al predecir el valor medio de viviendas (MEDV).

Las desviaciones estándar ($\pm$ 0.42 para MAE y $\pm$ 0.94 para RMSE) son moderadas, lo que indica que el rendimiento varía entre splits, pero no de manera incontrolada.

2. Ejercicio 2: Stroke Prediction Dataset