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]
```

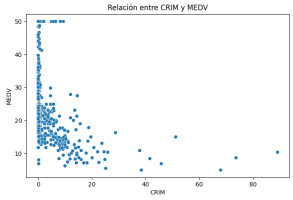
1.2. Objetivo

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

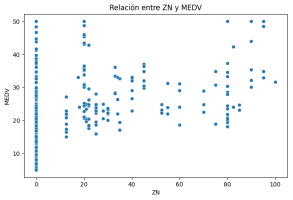
1.3. Tareas Requeridas

1.3.1. Implementación de Modelos

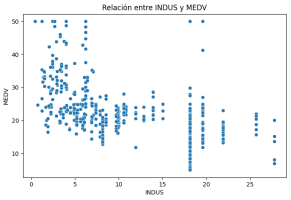
Antes de realizar los diferentes modelos, vamos a buscar significancia dentro de las variables para evitar utilizar datos que no aporten al trabajo.



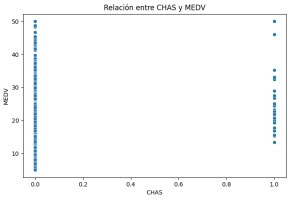
(a) Tasa de criminalidad per cápita



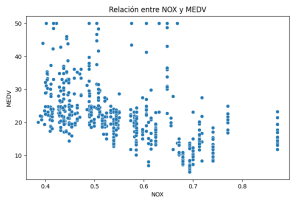
(b) Proporción de terrenos residenciales



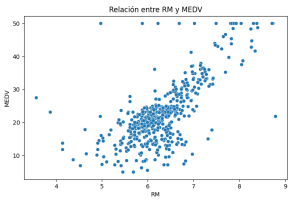
(c) Proporción de terrenos industriales



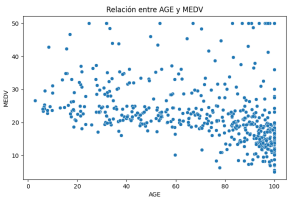
(d) Variable Dummy del Río Charles



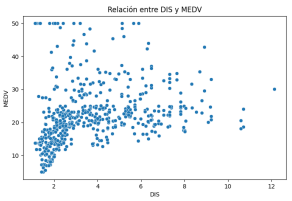
(e) Concentración de Óxidos de Nitrógeno



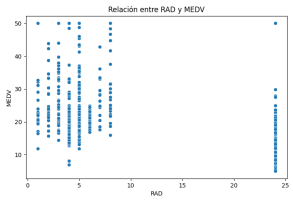
(f) Número promedio de habitaciones



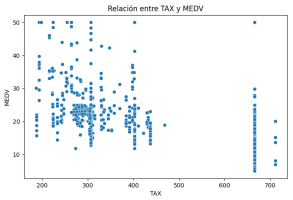
(g) Edad



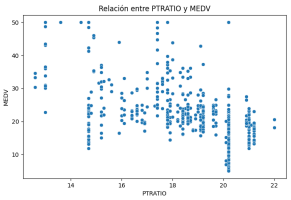
(h) Distancia a centros de empleo



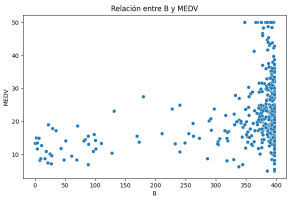
(i) Accesibilidad a autopistas radiales



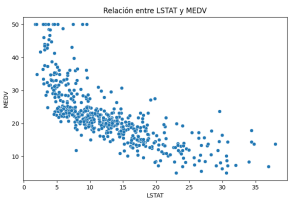
(j) Tasa de impuesto predial



(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. K-Nearest Neighbors (KNN)

Primero, definimos nuestras variables para poder trabajar con el modelo KNN.

```
x = muestra[['pregnancies', 'glucose', 'blood_pressure', '
    skin_thickness',
'insulin', 'bmi', 'diabetes_pedigree', 'age']]
y = muestra['outcome']

X_train, X_test, y_train, y_test = train_test_split(x, y,
    random_state=4181, shuffle=True)
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Ahora, vamos a calcular el valor K que maximice la exactitud del modelo.

```
    k_range = range(1, 21)
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    scores.append(knn.score(X_test, y_test))
plt.figure()
plt.xlabel('k')
plt.ylabel('Exactitud')
plt.scatter(k_range, scores)
plt.xticks([0, 5, 10, 15, 20])
plt.title('Exactitud vs. k en KNN')
plt.show()
```

Obteniendo la siguiente gráfica:

Figura 2: Exactitud vs. k en KNN

Por lo que decidimos utilizar el valor k=13 para nuestro modelo KNN.

```
knn = KNeighborsClassifier(n_neighbors=13)
knn.fit(X_train, y_train)
print("Exactitud en entrenamiento: {}".format(knn.score(X_train,
    y_train)))
print("Exactitud en prueba: {}".format(knn.score(X_test, y_test))
    )

y_pred = knn.predict(X_test)
y_pred_proba = knn.predict_proba(X_test)[:, 1]

precision = precision_score(y_test, y_pred)
sensibilidad = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

print(f'Precision: {precision}')
```

```
print(f'Sensibilidad: {sensibilidad}')
print(f'F1 Score: {f1}')
print(f'AUC: {auc}')
```

Obteniendo los siguientes resultados:

Cuadro 2: Métricas de Rendimiento del Modelo KNN

Métrica	**Valor**
Mejor valor de K	13
Exactitud en entrenamiento	77.3 %
Exactitud en prueba	72.8 %
Precisión (clase 1)	52.8 %
Sensibilidad	52.8 %
F1-Score	52.8 %
AUC	0.6683

1.5.1. Interpretación de Resultados

El modelo K-NN con k=13 demuestra un rendimiento moderado para la clasificación de diabetes, con una exactitud del 72.8 % en el conjunto de prueba. La diferencia entre la exactitud de entrenamiento (77.3 %) y prueba (72.8 %) sugiere un ligero sobreajuste, pero dentro de límites aceptables.

La precisión y sensibilidad idénticas del 52.8 % indican que el modelo mantiene un balance equilibrado entre la detección de verdaderos positivos y la minimización de falsos positivos. El valor de AUC de 0.668 muestra una capacidad discriminativa moderada, superior al azar pero con margen de mejora.

1.6. Linear Discriminant Analysis (LDA)

En la Linear Discriminant Analysis vamos a escalar nuevamente los datos antes de proceder:

```
    x = muestra[['pregnancies', 'glucose', 'blood_pressure', '
       skin_thickness',
'insulin', 'bmi', 'diabetes_pedigree', 'age']]
y = muestra['outcome']

x_train, x_test, y_train, y_test = train_test_split(x, y,
   test_size=0.2, random_state=4181, shuffle=True)

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

Ahora, procedemos a realizar el modelo LDA:

```
    lda = LinearDiscriminantAnalysis()
    lda.fit(x_train_scaled, y_train)
```

```
    y_pred_lda = lda.predict(x_test_scaled)
```

Obteniendo los siguientes resultados:

Cuadro 3: Métricas de Rendimiento del Modelo Linear Discriminant Analysis

Métrica	**Valor**
Exactitud en entrenamiento	77.75 %
Exactitud en prueba	75 %
Precisión (clase 1)	61.29 %
Sensibilidad	59.38 %
F1-Score	60.32 %
AUC	0.7086

1.6.1. Interpretación de Resultados - Linear Discriminant Analysis

El modelo LDA presenta un rendimiento sólido y equilibrado para la clasificación de diabetes, destacándose por su excelente capacidad de generalización con una diferencia mínima de 2.75 puntos porcentuales entre la exactitud de entrenamiento (77.75 %) y prueba (75 %), evidenciando prácticamente ausencia de sobreajuste. Con una precisión del 61.29 %, sensibilidad del 59.38 % y F1-Score del 60.32 %, LDA supera significativamente a KNN en todas las métricas (+8.5 puntos en precisión) y demuestra una capacidad discriminativa buena con un AUC de 0.7086, aproximándose al rendimiento de regresión logística (0.710). La estabilidad del modelo y su superior rendimiento en la minimización de falsos positivos lo posicionan como una opción robusta para la clasificación de diabetes, capturando efectivamente la estructura lineal de separación entre clases mientras mantiene un balance razonable entre precisión y sensibilidad.

1.7. Quadratic Discriminant Analysis (QDA)

Para realizar el Quadratic Discriminant Analysis utilizamos el siguiente código:

```
x = muestra[['pregnancies', 'glucose', 'blood_pressure', '
    skin_thickness',
'insulin', 'bmi', 'diabetes_pedigree', 'age']]
y = muestra['outcome']

x_train, x_test, y_train, y_test = train_test_split(x, y,
    test_size=0.2, random_state=4181, shuffle=True)

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

clf = QuadraticDiscriminantAnalysis()
clf.fit(x_train_scaled, y_train)

y_pred_qda = clf.predict(x_test_scaled)
```

Con este código obtenemos los siguientes resultados:

Cuadro 4: Métricas de Rendimiento del Modelo Quadratic Discriminant Analysis

Métrica	**Valor**
Exactitud en entrenamiento	77 %
Exactitud en prueba	72 %
Precisión (clase 1)	55.88 %
Sensibilidad	59.38 %
F1-Score	57.58 %
AUC	0.6866

1.7.1. Interpretación de Resultados - Quadratic Discriminant Analysis

El modelo QDA presenta un rendimiento intermedio para la clasificación de diabetes, con una diferencia de 5 puntos porcentuales entre la exactitud de entrenamiento (77 %) y prueba (72 %), indicando un ligero sobreajuste superior al observado en LDA pero comparable con KNN. Con una precisión del 55.88 %, sensibilidad del 59.38 % y F1-Score del 57.58 %, QDA se posiciona entre KNN y LDA en términos de rendimiento general, superando a KNN (+3 puntos en precisión) pero siendo inferior a LDA (-5.4 puntos en precisión). Su capacidad discriminativa con un AUC de 0.6866 resulta intermedia, superior a KNN (0.668) pero inferior tanto a LDA (0.709) como a regresión logística (0.710), sugiriendo que la flexibilidad adicional de las fronteras cuadráticas no se traduce en una mejora significativa del rendimiento para este conjunto de datos específico, donde las relaciones parecen ser predominantemente lineales.

1.8. Naive Bayes Gaussiano

Para realizar el Naive Bayes Gaussiano utilizamos el siguiente código:

```
x = muestra[['pregnancies', 'glucose', 'blood_pressure', '
    skin_thickness',
'insulin', 'bmi', 'diabetes_pedigree', 'age']]
y = muestra['outcome']

x_train, x_test, y_train, y_test = train_test_split(x, y,
    test_size=0.2, random_state=4181, shuffle=True)

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

nb = GaussianNB()
nb.fit(x_train_scaled, y_train)

y_pred_nb = nb.predict(x_test_scaled)
```

Con este código obtenemos los siguientes resultados:

Cuadro 5: Métricas de Rendimiento del Modelo Naive Bayes Gaussiano

Métrica	**Valor**
Exactitud en entrenamiento	77 %
Exactitud en prueba	79 %
Precisión (clase 1)	65.71 %
Sensibilidad	71.88 %
F1-Score	68.66 %
AUC	0.7711

1.8.1. Interpretación de Resultados - Naive Bayes Gaussiano

El modelo Naive Bayes Gaussiano presenta el mejor rendimiento general entre todos los modelos evaluados, destacándose por una situación excepcional donde la exactitud en prueba (79 %) supera a la de entrenamiento (77 %), indicando una excelente capacidad de generalización sin sobreajuste. Con la mayor sensibilidad (71.88 %) de todos los modelos, Naive Bayes demuestra superior capacidad para detectar casos reales de diabetes, lo cual es crítico en aplicaciones médicas donde no identificar un caso positivo puede tener consecuencias graves. Su precisión del 65.71 % es la más alta obtenida, minimizando efectivamente los falsos positivos, mientras que el F1-Score de 68.66 % refleja el mejor balance entre precisión y sensibilidad. El AUC de 0.7711 confirma la mejor capacidad discriminativa entre todas las técnicas evaluadas, superando incluso a la regresión logística (0.710) y LDA (0.709). Estos resultados sugieren que el supuesto de independencia condicional del algoritmo, aunque teóricamente restrictivo, funciona efectivamente en este conjunto de datos específico, posicionando a Naive Bayes como la opción más robusta para la clasificación de diabetes en términos de rendimiento predictivo y estabilidad del modelo.

1.9. 10-fold Cross-Validation

Para realizar nuestro Cross-Validation utilizamos el siguiente código:

```
x = muestra[['pregnancies', 'glucose', 'blood_pressure', 'skin_thickness',
        'insulin', 'bmi', 'diabetes_pedigree', 'age']]
y = muestra['outcome']

cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=4181)

scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']

scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)

#logistic Regression Cross Validation
logistic = LogisticRegression(penalty=None)
lr_results = cross_validate(logistic, x_scaled, y, cv=cv, scoring=scoring)

#K-Nearest Neighboor Cross Validation]
scaler_knn = MinMaxScaler()
x_scaled_knn = scaler_knn.fit_transform(x)
knn = KNeighborsClassifier(n_neighbors=13)
knn_results = cross_validate(knn, x_scaled, y, cv=cv, scoring=scoring)

#Linear Discriminant Analysys (LDA) Cross Validation

lda = LinearDiscriminantAnalysis()
lda_results = cross_validate(lda, x_scaled, y, cv=cv, scoring=scoring)

#Quadratic Discriminant Analysis (QDA) Cross Validation
clf = QuadraticDiscriminantAnalysis()
qda_results = cross_validate(clf, x_scaled, y, cv=cv, scoring=scoring)

#Naive Bayes Gaussiano Cross Validation
nb = GaussianNB()
nb_results = cross_validate(nb, x_scaled, y, cv=cv, scoring=scoring)
```

Obteniendo los siguientes resultados:

Cuadro 6: Resultados de Validación Cruzada 10-Fold

Modelo	Exactitud	Precisión	Sensibilidad	F1-Score	AUC
Reg. Logística	0.766±0.081	0.713±0.131	0.544±0.186	0.605±0.149	0.826±0.087
KNN (k=13)	0.748±0.059	0.676±0.093	0.514±0.158	0.575±0.123	0.782±0.093
LDA	0.762±0.090	0.705±0.147	0.538±0.199	0.597±0.161	0.823±0.092
QDA	0.744±0.107	0.659±0.162	0.538±0.195	0.584±0.176	0.798±0.097
Naive Bayes	0.768±0.093	0.687±0.142	0.613±0.186	0.639±0.148	0.809±0.098

1.9.1. Interpretación de Resultados - 10-fold Cross-Validation

La validación cruzada revela que, para las diferentes medidas, existen diferentes modelos que se desempeñan mejor.

- Exactitud: El modelo de Naive Bayes Gaussiano tiene la mayor exactitud promedio.
- Precisión: La regresión logística tiene la mayor precisión promedio.
- Sensibilidad: El Naive Bayes Gaussiano tiene la mayor sensibilidad promedio.
- F1-Score: El Naive Bayes Gaussiano tiene el mayor F
- AUC: La regresión logística tiene el mayor AUC promedio.

1.10. Discusión de Resultados

1.10.1. Contexto Médico y Relevancia Clínica

El dataset Pima Indians Diabetes presenta características particulares que influyen significativamente en la interpretación de los resultados obtenidos. Las mujeres de la tribu Pima tienen una de las tasas más altas de diabetes tipo 2 a nivel mundial, con una prevalencia que puede superar el 50 % en adultos, lo que convierte a esta población en un caso de estudio crítico para la predicción temprana de la enfermedad.

1.10.2. Análisis Comparativo de Modelos

Los resultados de validación cruzada revelan diferencias importantes entre los modelos evaluados, cada uno con implicaciones clínicas específicas:

Regresión Logística: El Estándar Clínico
La regresión logística demostró ser el modelo más robusto en términos de capacidad discriminativa (AUC = 0.826) y precisión (0.713), características fundamentales en el contexto médico. Su alta precisión minimiza los falsos positivos, reduciendo la ansiedad innecesaria en pacientes y los costos asociados con pruebas adicionales. El AUC superior indica una excelente capacidad para distinguir entre pacientes con y sin diabetes, lo cual es crítico para la toma de decisiones clínicas informadas.

Naive Bayes: Máxima Sensibilidad Clínica

El modelo Naive Bayes Gaussiano sobresale por su sensibilidad superior (0.613), característica crucial en medicina preventiva donde el costo de no detectar un caso de diabetes (falso negativo) puede ser devastador para la salud del paciente. En el contexto de la tribu Pima, donde la diabetes puede progresar rápidamente y causar complicaciones graves, maximizar la detección temprana es prioritario sobre minimizar los falsos positivos.

Implicaciones de los Modelos Lineales (LDA vs QDA)

La superioridad de LDA sobre QDA (AUC: 0.823 vs 0.798) sugiere que las relaciones entre las variables predictoras y el riesgo de diabetes en esta población son predominantemente lineales. Este hallazgo tiene implicaciones importantes para el entendimiento de la patofisiología de la diabetes en la población Pima, indicando que los factores de riesgo actúan de manera aditiva rather than interactive.

1.10.3. Consideraciones Específicas del Contexto Pima

Factores Genéticos y Ambientales

Los resultados confirman la importancia de factores modificables como el BMI y la glucosa, pero también revelan la influencia significativa de variables como pregnancies y diabetes pedigree, que reflejan tanto factores genéticos como ambientales específicos de esta población. La alta prevalencia de diabetes gestacional en las mujeres Pima se refleja en la importancia predictiva del número de embarazos.

1.10.4. Recomendaciones Clínicas Basadas en los Resultados

- **Screening Primario:** Utilizar Naive Bayes para maximizar la detección temprana en programas de screening poblacional
- **Confirmación Diagnóstica:** Emplear regresión logística cuando se requiera alta especificidad para confirmar casos sospechosos
- **Monitoreo Continuo:** Implementar LDA para seguimiento longitudinal de pacientes de riesgo, aprovechando su estabilidad y interpretabilidad

1.11. Matriz de Confusión

Figura 3: Embarazos

Según la Matriz de Confusión, podemos interpretar los resultados como:

- 79 de 100 casos positivos fueron correctamente identificados (Verdaderos Positivos).
- 12 falsos positivos fueron identificados, lo que indica que 12 personas sin diabetes fueron clasificadas erróneamente como diabéticas.
- 9 falsos negativos, lo que significa que 9 personas con diabetes no fueron detectadas por el modelo.

1.12. Conclusión

En el contexto específico de la población Pima Indians, donde la diabetes representa una crisis de salud pública, los modelos desarrollados ofrecen herramientas valiosas para la detección temprana y prevención. La elección del modelo óptimo debe basarse en el objetivo clínico específico: Naive Bayes para screening poblacional donde la sensibilidad es prioritaria, y regresión logística para confirmación diagnóstica donde la especificidad es crucial. La implementación exitosa de estos modelos requiere integración cuidadosa con el contexto cultural y los recursos de salud disponibles en la comunidad.

2. Ejercicio 2: California Housing

Se trabajará con el dataset California Housing, que contiene información sobre viviendas en California. El objetivo es predecir el valor mediano de las viviendas (variable objetivo continua, en unidades de 100,000 dólares).

Variables predictoras:

- MedInc: Ingreso mediano del hogar (en decenas de miles de dólares)
- HouseAge: Edad mediana de las viviendas (en años)
- AveRooms: Número promedio de habitaciones por vivienda
- AveBedrms: Número promedio de dormitorios por vivienda
- Population: Población del bloque
- AveOccup: Ocupación promedio de las viviendas
- Latitude: Latitud del bloque
- Longitude: Longitud del bloque

Para exportar los datos utilizamos el siguiente código:

```
from sklearn.datasets import fetch_california_housing
california = fetch_california_housing()
X = california.data
y = california.target
```

2.0.1. Búsqueda de Hiperparámetros con Grid Search:

Realizamos grid search con validación cruzada de 5 folds para encontrar los mejores hiperparámetros para los tres modelos:

```
cv = KFold(n_splits=5, shuffle=True, random_state=4181)
```

Lasso Regression:

```
lasso_params = {
    'alpha': np.logspace(-4, 2, 20)
}
lasso = Lasso(random_state=42, max_iter=10000)
lasso_grid = GridSearchCV(
    estimator=lasso,
    param_grid=lasso_params,
    cv=cv,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=1
)

lasso_grid.fit(X_train_scaled, y_train)
```

Ridge Regression:

```
ridge_params = {
    'alpha': np.logspace(-4, 2, 20)
}
ridge = Ridge(random_state=42)
ridge_grid = GridSearchCV(
    estimator=ridge,
    param_grid=ridge_params,
    cv=cv,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=1
)
ridge_grid.fit(X_train_scaled, y_train)
```

Elastic Net:

```
elastic_params = {
    'alpha': np.logspace(-4, 2, 10),
    'l1_ratio': np.linspace(0, 1, 10)
}

elastic = ElasticNet(random_state=42, max_iter=10000)
elastic_grid = GridSearchCV(
    estimator=elastic,
    param_grid=elastic_params,
    cv=cv,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=1
)
elastic_grid.fit(X_train_scaled, y_train)
```

2.0.2. Evaluación y Comparación de Modelo:

Para cada modelo, con sus mejores hiperparámetros encontramods evaluaremos usando cross validation de 5 folds en el conjunto de entrenamiento:

```
lasso_validation = cross_val_score(lasso_grid.best_estimator_,
    X_test_scaled, y_test, cv=5, scoring='neg_mean_squared_error')
ridge_validation = cross_val_score(ridge_grid.best_estimator_,
    X_test_scaled, y_test, cv=5, scoring='neg_mean_squared_error')
elastic_validation = cross_val_score(elastic_grid.best_estimator_
    , X_test_scaled, y_test, cv=5, scoring='neg_mean_squared_error'
    )
```

Dándonos como resultado:

Cuadro 7: Resultados de Validación Cruzada 5-Fold

Modelo	**CV Score**
Lasso	0.5028 (+/- 0.1239)
Ridge	0.5114 (+/- 0.1317)
ElasticNet	0.5026 (+/- 0.1225)

Ahora, vamos a calcular Root Mean Square Error y Mean Absolute Error para cada modelo:

```
best_lasso = lasso_grid.best_estimator_
best_ridge = ridge_grid.best_estimator_
best_elastic = elastic_grid.best_estimator_

y_pred_lasso = best_lasso.predict(X_test_scaled)
y_pred_ridge = best_ridge.predict(X_test_scaled)
y_pred_elastic = best_elastic.predict(X_test_scaled)
```

Cuadro 8: Resultados de RMSE y MAE

Modelo	**RMSE**	**MAE**
Lasso	0.7427	0.5365
Ridge	0.7433	0.5360
ElasticNet	0.7426	0.5366

2.0.3. Análisis de Resultados

Lasso:

¿Cuántas variables tienen coeficientes exactamente iguales a cero?
El modelo Lasso tiene 1 variable con coeficiente exactamente igual a cero.

¿Qué variables fueron eliminadas del modelo?
La variable eliminada por Lasso es Population.

Ridge:

¿Cuántas variables tienen coeficientes cercanos a cero (—coeficiente—¡0,01)?
Tenemos 1 variable con coeficiente cercano a cero.

¿Cómo se comparan las magnitudes de los coeficientes con los de Lasso?
Al sumar el total de los valores absolutos tenemos:

- Lasso: 3.2259
- Ridge: 3.3015

También, viendo los comportamientos por variables obtenemos:

Cuadro 9: Comparativa por Variable

Variable	**Lasso**	**Ridge**
MedInc	0.8295	0.8382
HouseAge	0.1249	0.1248
AveRooms	-0.2408	-0.2612
AveBedrms	0.2839	0.3039
Population	-0.0000	-0.0004
AveOccup	-0.0379	-0.0397
Latitude	-0.8686	-0.8803
Longitude	-0.8403	-0.8530

Ridge mantiene todas las variables en el modelo con coeficientes ligeramente superiores en magnitud (suma total de 3.3015 vs 3.2259 de Lasso), aplicando una penalización uniforme que reduce pero no elimina completamente ninguna variable. En contraste, Lasso demuestra su capacidad de selección automática de características al eliminar completamente la variable Population, asignándole un coeficiente de exactamente cero, mientras que Ridge la conserva con un valor mínimo de -0.0004. Las diferencias más pronunciadas se observan en AveRooms y AveBedrms, donde Ridge presenta coeficientes 8.48

Elastic Net:

Basado en el mejor valor de l1 ratio encontrado, ¿el modelo se comporta más como Lasso o como Ridge?

El modelo se comporta más como Lasso.

¿Qué ventajas ofrece Elastic Net sobre los otros dos métodos en este dataset?

En este caso particular, Elastic Net no ofrece ventajas significativas sobre Lasso y Ridge debido a que converge a un comportamiento idéntico al de Lasso puro (l1_ratio = 1.0). Esto significa que el algoritmo de optimización determinó que la regularización L1 pura era la más efectiva para este dataset, eliminando completamente el componente L2. Sin embargo, las ventajas teóricas de Elastic Net incluyen:

- Estabilidad en selección de variables: cuando existen grupos de variables altamente correlacionadas, Elastic Net tiende a seleccionar grupos completos en lugar de elegir arbitrariamente una variable del grupo como hace Lasso.
- Flexibilidad en regularización: permite encontrar el balance óptimo entre selección de características (L1) y reducción uniforme de coeficientes (L2) mediante el parámetro l1_ratio.
- Robustez numérica: es menos sensible a pequeños cambios en los datos comparado con Lasso puro, especialmente cuando n ¡p (más variables que observaciones).

No obstante, en el dataset California Housing, la ausencia de multicolinealidad severa y la naturaleza de las variables explicativas hicieron que Lasso fuera suficiente, por lo que Elastic Net no proporcionó beneficios adicionales, convergiendo naturalmente hacia la regularización L1 pura y produciendo resultados prácticamente idénticos a Lasso tanto en coeficientes como en métricas de rendimiento (RMSE: 0.7426 vs 0.7427).

2.0.4. Conclusión

Según el Cuadro 9: Resultados de RMSE y MAE, podemos presentar que el mejor modelo es Elastic Net debido a:

- Tiene el menor RMSE (0.7426) lo que indica mejor capacidad predictiva
- Las diferencias entre los tres modelos son muy pequeñas
- Converge a Lasso puro, pero el proceso de optimización le permite encontrar una solución ligeramente superior.
- Ofrece mayor estabilidad numérica en el proceso de Optimización
- Elimina automáticamente variables irrelevantes al igual que Lasso.

En términos prácticos, los tres modelos tienen rendimientos idénticos, pero Elastic Net logra la mejor combinación de presición predictiva y selección automática de características, posicionándolo como la opción óptima para este dataset.

3. Referencias

Referencias

[1] Ciencia de Datos. *Regresión Logística con Python*. https://cienciadedatos.net/documentos/py17-regresion-logistica-python

[2] Aprende Machine Learning. *Clasificar con K-Nearest Neighbor: Ejemplo en Python*. https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/

[3] GeeksforGeeks. *Linear Discriminant Analysis in Machine Learning*. https://www.geeksforgeeks.org/machine-learning/ml-linear-discriminant-analysis/

[4] Scikit-learn Documentation. *QuadraticDiscriminantAnalysis*. https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html

[5] GeeksforGeeks. *Gaussian Naive Bayes*. https://www.geeksforgeeks.org/machine-learning/gaussian-naive-bayes/

[6] GeeksforGeeks. *Cross Validation using K-Fold with Scikit Learn*. https://www.geeksforgeeks.org/machine-learning/cross-validation-using-k-fold-with-scikit-learn/

[7] GeeksforGeeks. *How to Calculate Mean Absolute Error in Python*. https://www-geeksforgeeks-org.translate.goog/python/how-to-calculate-mean-absolute-error-in-python/