Taller de Aprendizaje Automático

Proyecto de aprendizaje automático

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Desarrollo de un proyecto de punta a punta

- diversas dificultades y errores comunes
- varias metodologías*† (p. ej. CRISP-DM)
- en general involucran:
  - entender el contexto y el problema
  - entender y preparar los datos
  - entrenar y seleccionar modelos
  - evaluar y generar la solución
  - desplegar, monitorear y mantener
- seguiremos la propuesta del libro‡

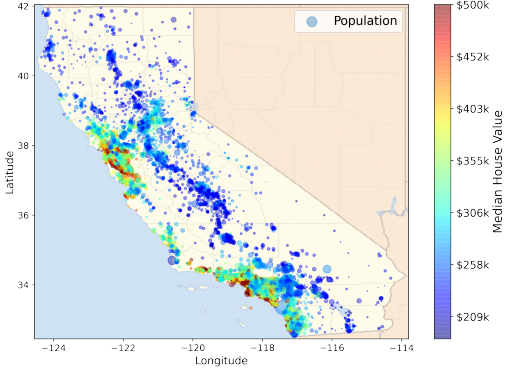
Cross-industry Standard Process for Data Mining (CRISP-DM)

*L. A. Kurgan and P. Musilek, "A survey of knowledge discovery and data mining process models," *Knowledge Engineering Review*, vol. 21, p. 1–24, mar 2006

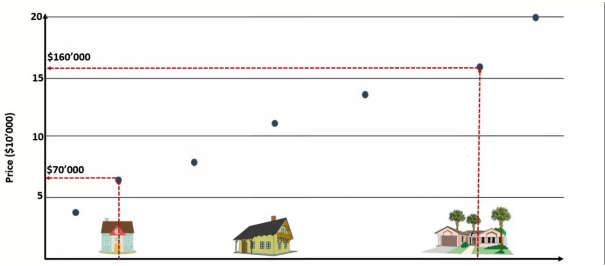
†S. Sharma, K.-M. Osei-Bryson, and G. M. Kasper, "Evaluation of an integrated knowledge discovery and data mining process model," *Expert Systems with Applications*, vol. 39, no. 13, pp. 11335–11348, 2012

‡A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition*. O'Reilly Media, Inc., 2022

Entender el problema



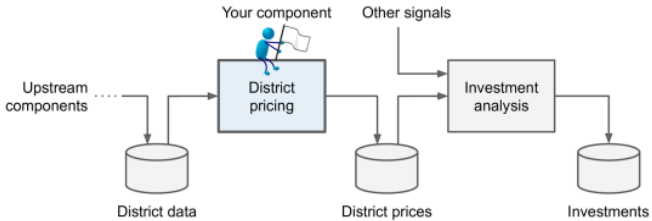
Datos: Censo de California por distrito (1990)



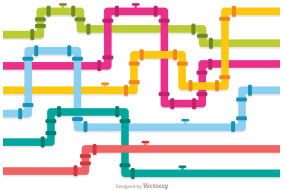
Tarea: Predecir valor medio de propiedades por distrito.

- aprendizaje supervisado
- regresión (múltiple y univariada)

Entender el problema



Tubería (pipeline) de aprendizaje automático para el problema.



Tubería

- secuencia de componentes de procesamiento
- la interfaz entre componentes son los datos

Entender el problema

- seleccionar una medida de desempeño

  Root Mean Square Error (ℓ_2): $\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(\mathbf{x}^{(i)}) - y^{(i)})^2}$

  Mean Absolute Error (ℓ_1): $\mathrm{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}|h(\mathbf{x}^{(i)}) - y^{(i)}|$

- informarse sobre la solución actual (si existe)
  - se usa estimación manual por parte de expertos
  - que con frecuencia tiene errores mayores al 20%

- verificar los supuestos
  - el valor estimado de las propiedades no es categórico (p.ej. *alto*, *medio*, *bajo*)

Descargar y cargar los datos

- sugerencia: crear funciones para descargar y cargar los datos

```
import os
import tarfile
import urllib.request

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

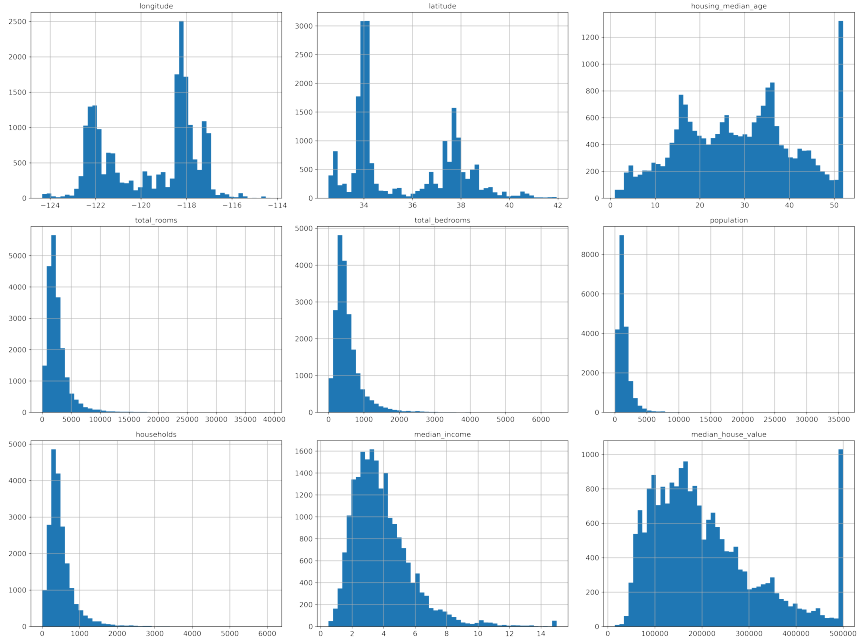
Descargar y cargar los datos

```
import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

```
housing = load_housing_data()
housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

```
housing.info()
housing.describe()
housing["ocean_proximity"].value_counts()
```


longitude
latitude
housing_median_age
total_rooms
total_bedrooms
population
households
median_income
median_house_value

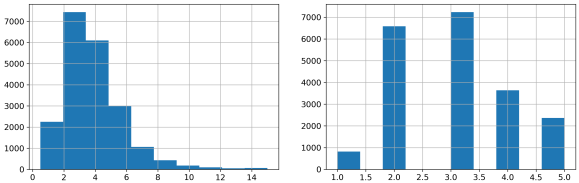
Crear el conjunto de test

```
import numpy as np

def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]


train_set, test_set = split_train_test(housing, 0.2)



from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

- sugerencia: agregar identificador de cada instancia ¿para qué? ¿de qué forma?

Crear el conjunto de test

- muestreo estratificado: asegurarse que el conjunto de test sea representativo




```
from sklearn.model_selection import StratifiedShuffleSplit

housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

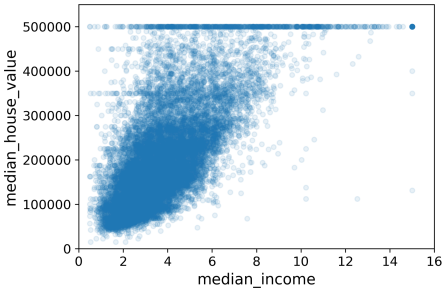
Buscar correlaciones

correlacion de Pearson

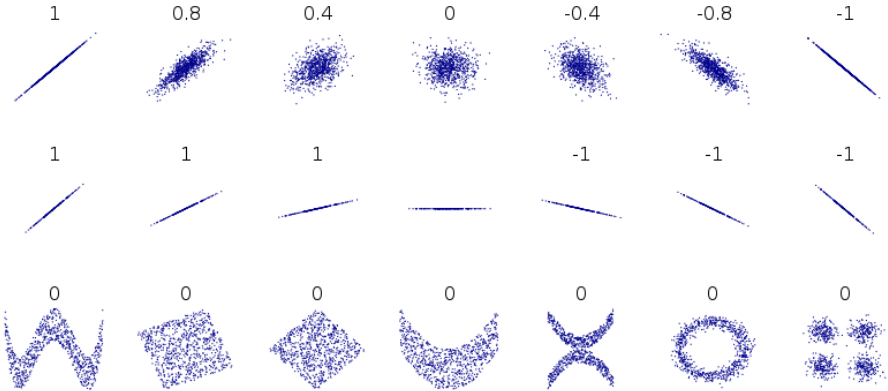
```
corr_matrix = housing.corr()

corr_matrix["median_house_value"].sort_values()
```

```
median_house_value    1.000000
median_income         0.687160
total_rooms           0.135097
housing_median_age    0.114110
households            0.064506
total_bedrooms        0.047689
population           -0.026920
longitude            -0.047432
latitude             -0.142724
```

$$r(u, v) = \frac{\sum_i (u_i - \bar{u})(v_i - \bar{v})}{\sqrt{\sum_i (u_i - \bar{u})^2} \sqrt{\sum_i (v_i - \bar{v})^2}}$$




```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income",
              "total_rooms", "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
save_fig("scatter_matrix_plot")
```


1
0.8
0.4
0
-0.4
-0.8
-1
1
1
1
-1
-1
-1
0
0
0
0
0
0
0

Combinar atributos

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]

corr_matrix = housing.corr()

corr_matrix["median_house_value"].sort_values()

median_house_value          1.000000
median_income               0.687160
rooms_per_household         0.146285
total_rooms                 0.135097
housing_median_age          0.114110
households                  0.064506
total_bedrooms              0.047689
population_per_household   -0.021985
population                 -0.026920
longitude                  -0.047432
latitude                   -0.142724
bedrooms_per_room          -0.259984
```

Preparar los datos

- sugerencia: crear funciones para preparar los datos
  - facilita reproducibilidad sobre nuevos datos
  - facilita probar diferentes transformaciones

- datos faltantes: `total_bedrooms`

```
housing.dropna(subset=["total_bedrooms"])    # option 1
housing.drop("total_bedrooms", axis=1)       # option 2
median = housing["total_bedrooms"].median()  # option 3
housing["total_bedrooms"].fillna(median, inplace=True)

from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
X = imputer.transform(housing_num)
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing.index)
```

Atributos categóricos

```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(10)
```

```
  ocean_proximity
17606  <1H OCEAN
18632  <1H OCEAN
14650  NEAR OCEAN
3230   INLAND
3555   <1H OCEAN
19480  INLAND
8879   <1H OCEAN
13685  INLAND
4937   <1H OCEAN
4861   <1H OCEAN
```

```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encoded[:10]
```

```
from sklearn.preprocessing import OneHotEncoder

cat_enc = OneHotEncoder()
housing_cat_1hot = cat_enc.fit_transform(housing_cat)
housing_cat_1hot.toarray()
```

```
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       ...,
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

Transformación a medida

```
from sklearn.base import BaseEstimator, TransformerMixin

# column index
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

Pipeline de transformaciones

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)


from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

Entrenar y evaluar con los datos de entramiento

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)


from sklearn.metrics import mean_squared_error

housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse) % 68628.19819848923


from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)


housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse) % 0.0
```

Evaluando con validación cruzada

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)

display_scores(tree_rmse_scores)
```

```
  Mean: 71407.68766037929
  Standard deviation: 2439.4345041191004
```

```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
  Mean: 69052.46136345083
  Standard deviation: 2731.674001798344
```

Ajuste de hiperparámetros

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

param_grid = [
    # try 12 (3×4) combinations of hyperparameters
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    # then try 6 (2×3) combinations with bootstrap set as False
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
# train across 5 folds, that's a total of (12+6)*5=90 rounds of training
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)

>>> grid_search.best_params_
{'max_features': 8, 'n_estimators': 30}
```

Analizar los modelos y sus errores

```
feature_importances = grid_search.best_estimator_.feature_importances_

[(0.36615898061813423, 'median_income'),
 (0.16478099356159054, 'INLAND'),
 (0.10879295677551575, 'pop_per_hhold'),
 (0.07334423551601243, 'longitude'),
 (0.06290907048262032, 'latitude'),
 (0.056419179181954014, 'rooms_per_hhold'),
 (0.053351077347675815, 'bedrooms_per_room'),
 (0.04114379847872964, 'housing_median_age'),
 (0.014874280890402769, 'population'),
 (0.014672685420543239, 'total_rooms'),
 ...
 (6.0280386727366e-05, 'ISLAND')]
```

- sugerencia: analizar los errores que se cometen
  - ¿por qué se cometen?
  - ¿cómo podrían solucionarse?

Evaluar en el conjunto de test

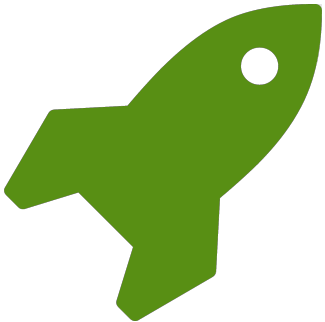
```
final_model = grid_search.best_estimator_

X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)

>>> final_rmse
47730.22690385927
```

Presentar la solución

- documentar el proceso
- ¿la solución logra su objetivo?
- ¿qué funcionó y qué no?
- enumerar supuestos y limitaciones
- identificar los aspectos clave
- generar una demostración

Desplegar y monitorear

- pulir código, documentación, tests
- encapsular el modelo y el pipeline
- web service, cloud platform
- escribir código de monitoreo
- recolectar nuevos datos regularmente
- volver a entrenar, refinar, evaluar
- crear y gestionar respaldos



Actividades

Leer el Capítulo 2 y correr el resto del notebook.

En particular:

- probar con SVM (`sklearn.svm.SVR`)
- remplazar `GridSearchCV` por `RandomizedSearchCV`
- hacer un solo pipeline que incluya la predicción

Responder cuestionario EVA sobre `GridSearchCV` vs `RandomizedSearchCV`.

Referencias

L. A. Kurgan and P. Musilek, "A survey of knowledge discovery and data mining process models," *Knowledge Engineering Review*, vol. 21, p. 1–24, mar 2006.

S. Sharma, K.-M. Osei-Bryson, and G. M. Kasper, "Evaluation of an integrated knowledge discovery and data mining process model," *Expert Systems with Applications*, vol. 39, no. 13, pp. 11335–11348, 2012.

A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition*. O'Reilly Media, Inc., 2022.

Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning From Data*. AMLBook, 2012.

A. Halevy, P. Norvig, and F. Pereira, "The unreasonable effectiveness of data," *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8–12, 2009.

M. Banko and E. Brill, "Scaling to very very large corpora for natural language disambiguation," in *Proceedings of the 39th ACL 2001*, 2001.