Week_2_End_to_End_Machine_Learning_Project

March 11, 2024

1 Week 2 - End-to-end Machine Learning Project

1.1 Download the Data

[ ]:
```
from pathlib import Path
import pandas as pd
import numpy as np

def load_housing_data():
    filepath = "C:/Users/chest/OneDrive - Universiti Malaya/UM/Teaching/KIG4068␣
 ↪Machine Learning/Data/"
    return pd.read_csv(Path(filepath+"datasets/housing/housing.csv"))

housing = load_housing_data()
```

1.2 A Quick Look at the Data Structure

[ ]:
```
housing.head()
```

[ ]:
```
housing.info()
```

[ ]:
```
housing.describe()
```

[ ]:
```
import matplotlib.pyplot as plt

plt.rc('font', size=14)
plt.rc('axes', labelsize=14, titlesize=14)
plt.rc('legend', fontsize=14)
plt.rc('xtick', labelsize=10)
plt.rc('ytick', labelsize=10)

housing.hist(bins=50, figsize=(12, 8))
plt.show()
```

1.3 Create a Test Set

[ ]:
```
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])

from sklearn.model_selection import train_test_split
strat_train_set, strat_test_set = train_test_split(
housing, test_size=0.2, stratify=housing["income_cat"], random_state=42)

for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

1.4 Explore and Visualise the Data to Gain Insights

1.4.1 Make a copy of training set for data exploration

[ ]:
```
housing = strat_train_set.copy()
```

1.4.2 Geographical scatter plot of the data

[ ]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True)
plt.show()
```

1.4.3 Better visualisation that highlights high-density areas

[ ]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True, alpha=0.2)
plt.show()
```

1.4.4 Better visualisation that highlights house value

[ ]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True,
s=housing["population"] / 100, label="population",
c="median_house_value", cmap="jet", colorbar=True,
legend=True, sharex=False, figsize=(10, 7))
plt.show()
```

1.4.5 Look for correlation

[ ]:
```
corr_matrix = housing.corr(numeric_only=True)
corr_matrix["median_house_value"].sort_values(ascending=False)

from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",
"housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
plt.show()
```

1.4.6 Look into more details on the median income vs median house value

[ ]:
```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
alpha=0.1, grid=True)
plt.show()
```

1.4.7 Experiment with Attribute Combinations

[ ]:
```
housing["rooms_per_house"] = housing["total_rooms"] / housing["households"]
housing["bedrooms_ratio"] = housing["total_bedrooms"] / housing["total_rooms"]
housing["people_per_house"] = housing["population"] / housing["households"]
```

1.4.8 Compare with previous correlation matrix

[ ]:
```
corr_matrix = housing.corr(numeric_only=True)
corr_matrix["median_house_value"].sort_values(ascending=False)
```

[ ]:
```
from pandas.plotting import scatter_matrix
attributes = [ "median_income", "rooms_per_house", "bedrooms_ratio"]
scatter_matrix(housing[attributes], figsize=(12, 8))
plt.show()
```

1.5 Prepare the Data for Machine Learning Algorithms

1.5.1 Split the features and target - get the target into its own dataframe

[ ]:
```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

1.5.2 Data cleaning

Look at the rows with NaN value

[ ]:
```
null_rows_idx = housing.isnull().any(axis=1)
housing.loc[null_rows_idx].head()
```

fill the missing value in total_bedrooms using Imputer function

[ ]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")

housing_num = housing.select_dtypes(include=[np.number]) #create new df, only
 ↪include numerical
imputer.fit(housing_num)
X = imputer.transform(housing_num)

housing_tr = pd.DataFrame(X, columns=housing_num.columns,
index=housing_num.index)
housing_tr.loc[null_rows_idx].head()
```

Handling Text and Categorical Attributes

[ ]:
```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(8)
```

Use OneHotEncoder

[ ]:
```
from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

1.6 Feature Scaling and Transformation

1.6.1 Option 1 - to use MinMaxScaler

[ ]:
```
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler(feature_range=(-1, 1))
housing_num_min_max_scaled = min_max_scaler.fit_transform(housing_num)
```

1.6.2 Option 2 - to use StandardScaler

[ ]:
```
from sklearn.preprocessing import StandardScaler
std_scaler = StandardScaler()
housing_num_std_scaled = std_scaler.fit_transform(housing_num)
```

1.6.3 Build Pipeline

[ ]:
```
from sklearn.pipeline import make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import make_column_selector

def ratio_pipeline():
    return make_pipeline(
        SimpleImputer(strategy="median"),
        FunctionTransformer(column_ratio, feature_names_out=ratio_name),
        StandardScaler())

def column_ratio(X):
    return X[:, [0]] / X[:, [1]]

def ratio_name(function_transformer, feature_names_in):
    return ["ratio"] # feature names out

def log_pipeline():
    return make_pipeline(
        SimpleImputer(strategy="median"),
        FunctionTransformer(np.log, feature_names_out="one-to-one"),
```

```
        StandardScaler())

def cat_pipeline():
    return make_pipeline(
        SimpleImputer(strategy="most_frequent"),
        OneHotEncoder(handle_unknown="ignore"))

def default_num_pipeline():
    return make_pipeline(
        SimpleImputer(strategy="median"),
        StandardScaler())

preprocessing = ColumnTransformer([
("bedrooms", ratio_pipeline(),
 ["total_bedrooms", "total_rooms"]),
("rooms_per_house", ratio_pipeline(),
 ["total_rooms", "households"]),
("people_per_house", ratio_pipeline(),
 ["population", "households"]),
("log", log_pipeline(),
 ["total_bedrooms", "total_rooms", "population",
"households", "median_income"]),
("cat", cat_pipeline(),
 make_column_selector(dtype_include=object)),
],remainder=default_num_pipeline())
# remaining col: housing_median_age

housing_prepared = preprocessing.fit_transform(housing)
housing_prepared.shape
```

1.7 Select and Train a Model

1.7.1 Example 1 - Linear Regression Model

[ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import root_mean_squared_error
lin_reg = make_pipeline(preprocessing, LinearRegression())
lin_reg.fit(housing, housing_labels)
housing_predictions = lin_reg.predict(housing)
lin_rmse = mean_squared_error(housing_labels, housing_predictions,squared=False)

lin_rmse
```

1.7.2 Example 2 - Decision Tree Model

```
[ ]: from sklearn.tree import DecisionTreeRegressor
tree_reg = make_pipeline(preprocessing, DecisionTreeRegressor(random_state=42))
tree_reg.fit(housing, housing_labels)
housing_predictions = tree_reg.predict(housing)
tree_rmse = mean_squared_error(housing_labels,␣
 ↪housing_predictions,squared=False)

tree_rmse
```

1.7.3 Example 3 - Random Forest Regressor

```
[ ]: from sklearn.ensemble import RandomForestRegressor
forest_reg = make_pipeline(preprocessing,
                           RandomForestRegressor(random_state=42))
forest_reg.fit(housing, housing_labels)
housing_predictions = forest_reg.predict(housing)
forest_rmse = mean_squared_error(housing_labels,␣
 ↪housing_predictions,squared=False)

forest_rmse
```

1.7.4 Cross Validation

```
[ ]: from sklearn.model_selection import cross_val_score
```

1.7.5 Cross Validate Example 1

```
[ ]: lin_rmses = -cross_val_score(lin_reg, housing, housing_labels,
scoring="neg_root_mean_squared_error", cv=10)
print("Cross-validation RMSEs (Kfold):", lin_rmses)
```

```
[ ]: from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
lin_rmses = -cross_val_score(lin_reg, housing, housing_labels,
                             scoring="neg_root_mean_squared_error", cv=skf)
print("Cross-validation RMSEs (Stratified Kfold):", lin_rmses)
```

1.7.6 Cross Validate Example 2

```
[ ]: from sklearn.model_selection import cross_val_score
tree_rmses = -cross_val_score(tree_reg, housing, housing_labels,
scoring="neg_root_mean_squared_error", cv=10)
print("Cross-validation RMSEs (Kfold):", tree_rmses.mean())
```

```
[ ]: from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
tree_rmses = -cross_val_score(tree_reg, housing, housing_labels,
                              scoring="neg_root_mean_squared_error", cv=skf)
print("Cross-validation RMSEs (Stratified Kfold):", tree_rmses.mean())
```

1.7.7 Cross Validate Example 3

```
[ ]: forest_rmses = -cross_val_score(forest_reg, housing, housing_labels,
scoring="neg_root_mean_squared_error", cv=10)
```

1.8 Model Fine-tuning

```
[ ]: from sklearn.pipeline import Pipeline
full_pipeline = Pipeline([
    ("preprocessing", preprocessing),
    ("random_forest", RandomForestRegressor(random_state=42)),
])
```

1.8.1 Option 1 (Grid Search)

```
[ ]: from sklearn.model_selection import GridSearchCV

param_grid = [
{'random_forest__max_features': [2, 4, 6, 8, 10, 12, 14, 16]}
]

grid_search = GridSearchCV(full_pipeline, param_grid, cv=3,
scoring='neg_root_mean_squared_error')

grid_search.fit(housing, housing_labels)
final_model = grid_search.best_estimator_

cv_res = pd.DataFrame(grid_search.cv_results_)
cv_res.sort_values(by="mean_test_score", ascending=False, inplace=True)
best_hyperparameters_grid = grid_search.best_params_
best_max_features = best_hyperparameters_grid['random_forest__max_features']


feature_importances = final_model["random_forest"].feature_importances_
feature_importances.round(2)

sorted(zip(feature_importances,
           final_model["preprocessing"].get_feature_names_out()),
       reverse=True)
```

1.8.2 Option 2 (Randomised Search)

[ ]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {'random_forest__max_features': randint(low=2, high=20)}

rnd_search = RandomizedSearchCV(
full_pipeline, param_distributions=param_distribs, n_iter=10, cv=3,
scoring='neg_root_mean_squared_error', random_state=42)

rnd_search.fit(housing, housing_labels)
final_model2 = rnd_search.best_estimator_

cv_res2 = pd.DataFrame(rnd_search.cv_results_)
cv_res2.sort_values(by="mean_test_score", ascending=False, inplace=True)
best_hyperparameters_rnd = rnd_search.best_params_
best_max_features = best_hyperparameters_rnd['random_forest__max_features']


feature_importances = final_model2["random_forest"].feature_importances_
feature_importances.round(2)

sorted(zip(feature_importances,
           final_model2["preprocessing"].get_feature_names_out()),
       reverse=True)
```

1.8.3 Evaluate Model on Test Set

[ ]:
```
print("Random Forest Grid")
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()
final_predictions = final_model.predict(X_test)
final_rmse = mean_squared_error(y_test, final_predictions, squared=False)
print(final_rmse)
```

[ ]:
```
print("Random Forest Randomized")
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()
final_predictions = final_model2.predict(X_test)
final_rmse = mean_squared_error(y_test, final_predictions, squared=False)
print(final_rmse)
```

[ ]:
```
from scipy import stats
confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=squared_errors.mean(),
```

```
                                 scale=stats.sem(squared_errors)))
```

[ ]:

```

```