Chapter 2: End-to-End Machine Learning Project

Summary from "Hands-On Machine Learning"

Study Notes

June 29, 2025

Contents

1 Chapter Overview **2**

2 The 8-Step ML Project Workflow **2**

3 Step 1: Frame the Problem **2**
- 3.1 Problem Definition 2
- 3.2 Performance Measures 3

4 Step 2: Get the Data **3**
- 4.1 Data Loading 3
- 4.2 Train/Test Split 3

5 Step 3: Explore the Data **4**
- 5.1 Geographic Visualization 4
- 5.2 Correlation Analysis 4

6 Step 4: Prepare the Data **4**
- 6.1 Handle Missing Values 4
- 6.2 Feature Scaling 4
- 6.3 Feature Engineering 5

7 Step 5: Select and Train Models **5**
- 7.1 Model Comparison 5
- 7.2 Cross-Validation 6

8 Step 6: Fine-Tune the Model **6**
- 8.1 Grid Search 6
- 8.2 Feature Importance 7

9 Step 7: Evaluate on Test Set **7**

10 Complete Pipeline **7**

11 Key Takeaways **8**

12 Common Pitfalls **8**

13 Conclusion **8**

1 Chapter Overview

This chapter demonstrates the complete workflow of a machine learning project using the California housing dataset. It covers all essential steps from problem framing to model deployment.

Key Learning Objectives:

- Understand the complete ML project workflow
- Learn data exploration and visualization techniques
- Master data preprocessing and feature engineering
- Compare multiple ML algorithms systematically
- Implement proper model evaluation and validation

2 The 8-Step ML Project Workflow

1. **Frame the Problem** - Define objectives and constraints
2. **Get the Data** - Collect and load the dataset
3. **Explore the Data** - Visualize and understand patterns
4. **Prepare the Data** - Clean and preprocess
5. **Select Models** - Try different algorithms
6. **Fine-tune** - Optimize hyperparameters
7. **Present Solution** - Communicate results
8. **Deploy & Monitor** - Production deployment

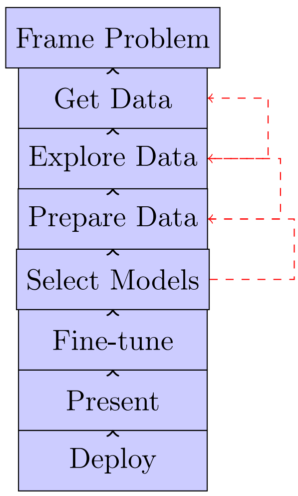

Figure 1: End-to-End Machine Learning Project Workflow

3 Step 1: Frame the Problem

3.1 Problem Definition

The California housing project predicts median house values using features like population, median income, and location.

Key Questions:

- What is the business objective?
- How will the solution be used?
- Supervised, unsupervised, or reinforcement learning?
- Classification or regression?
- How to measure performance?

3.2 Performance Measures

For regression problems:

$$\text{RMSE} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2} \tag{1}$$

$$\text{MAE} = \frac{1}{m}\sum_{i=1}^{m}|h(x^{(i)}) - y^{(i)}| \tag{2}$$

4 Step 2: Get the Data

4.1 Data Loading

Listing 1: Loading Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load the data
housing = pd.read_csv('housing.csv')

# Quick exploration
print(housing.head())
print(housing.info())
print(housing.describe())
```

4.2 Train/Test Split

Listing 2: Data Splitting

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit

# Simple random split
train_set, test_set = train_test_split(
    housing, test_size=0.2, random_state=42)

# Stratified split (better for small datasets)
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]
```

5 Step 3: Explore the Data

5.1 Geographic Visualization

California housing data shows clear geographic patterns with coastal areas having higher prices.

5.2 Correlation Analysis

Listing 3: Correlation Analysis

```python
# Compute correlation matrix
corr_matrix = housing.corr()

# Focus on target variable
print(corr_matrix["median_house_value"].sort_values(ascending=False))

# Scatter matrix for key attributes
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income",
              "total_rooms", "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
```

6 Step 4: Prepare the Data

6.1 Handle Missing Values

Listing 4: Missing Data Strategies

```python
# Three options:
# 1. Drop rows: housing.dropna(subset=["total_bedrooms"])
# 2. Drop column: housing.drop("total_bedrooms", axis=1)
# 3. Fill values: housing["total_bedrooms"].fillna(median)

# Using sklearn
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)
```

6.2 Feature Scaling

Two main approaches:

$$\text{Min-Max:} \quad x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3}$$

$$\text{Standardization:} \quad x_{scaled} = \frac{x - \mu}{\sigma} \tag{4}$$

Listing 5: Feature Scaling

```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Standardization
std_scaler = StandardScaler()
housing_scaled = std_scaler.fit_transform(housing_num)
```

```
# Min-Max scaling
minmax_scaler = MinMaxScaler()
housing_scaled = minmax_scaler.fit_transform(housing_num)
```

6.3 Feature Engineering

Listing 6: Creating New Features

```
# Combination attributes
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"]

# Custom transformer
from sklearn.base import BaseEstimator, TransformerMixin

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True):
        self.add_bedrooms_per_room = add_bedrooms_per_room

    def fit(self, X, y=None):
        return self

    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]
```

7 Step 5: Select and Train Models

7.1 Model Comparison

Table 1: Algorithm Performance Comparison

Model	RMSE	Speed
Linear Regression	68,628	Fast
Decision Tree	0 (overfitting)	Medium
Random Forest	49,682	Slow
SVM	111,094	Very Slow

Listing 7: Training Multiple Models

```python
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Linear Regression
lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, predictions)
lin_rmse = np.sqrt(lin_mse)

# Random Forest
forest_reg = RandomForestRegressor()
forest_reg.fit(housing_prepared, housing_labels)
```

7.2 Cross-Validation

Listing 8: K-Fold Cross-Validation

```python
from sklearn.model_selection import cross_val_score

def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

# 10-fold cross-validation
forest_scores = cross_val_score(forest_reg, housing_prepared,
    housing_labels,
                               scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

8 Step 6: Fine-Tune the Model

8.1 Grid Search

Listing 9: Hyperparameter Tuning

```python
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2,
        3, 4]},
]

forest_reg = RandomForestRegressor()
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                          scoring='neg_mean_squared_error')
grid_search.fit(housing_prepared, housing_labels)

print("Best parameters:", grid_search.best_params_)
```

8.2 Feature Importance

Random Forest can tell us which features are most important:

Listing 10: Feature Importance Analysis

```
feature_importances = grid_search.best_estimator_.feature_importances_
extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
cat_encoder = full_pipeline.named_transformers_["cat"]
cat_one_hot_attribs = list(cat_encoder.categories_[0])
attributes = num_attribs + extra_attribs + cat_one_hot_attribs
print(sorted(zip(feature_importances, attributes), reverse=True))
```

Top features typically include:

1. Median income (0.334)
2. Ocean proximity (0.364 combined)
3. Longitude/Latitude (0.088/0.069)
4. Rooms per household (custom feature)

9 Step 7: Evaluate on Test Set

Listing 11: Final Evaluation

```
# Use best model from grid search
final_model = grid_search.best_estimator_

# Prepare test data
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_predictions = final_model.predict(X_test_prepared)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)

print(f"Final RMSE: {final_rmse}")

# Confidence interval
from scipy import stats
squared_errors = (final_predictions - y_test) ** 2
confidence_interval = np.sqrt(stats.t.interval(
    0.95, len(squared_errors) - 1,
    loc=squared_errors.mean(),
    scale=stats.sem(squared_errors)))
```

10 Complete Pipeline

Listing 12: Full Processing Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
```

```python
from sklearn.preprocessing import OneHotEncoder

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

# Complete pipeline
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

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

# Transform data
housing_prepared = full_pipeline.fit_transform(housing)
```

11 Key Takeaways

Best Practices:

1. Create test set before exploring data
2. Use stratified sampling for small datasets
3. Explore data with visualizations
4. Handle missing values appropriately
5. Scale features for distance-based algorithms
6. Try multiple algorithms
7. Use cross-validation for evaluation
8. Fine-tune hyperparameters systematically
9. Analyze feature importance
10. Evaluate on test set only once

12 Common Pitfalls

Avoid These Mistakes:

- Looking at test data early
- Data snooping during tuning
- Ignoring data leakage
- Not handling categorical variables
- Forgetting feature scaling
- Overfitting to validation set
- Ignoring business context

13 Conclusion

Chapter 2 provides a systematic framework for ML projects. The key insight is that machine learning involves much more than just algorithms—proper data handling, systematic evaluation, and following a structured methodology are crucial for success.

The California housing example demonstrates how to:

- Frame problems correctly
- Handle real-world data issues
- Compare algorithms fairly
- Tune models systematically
- Evaluate performance rigorously

This workflow can be adapted to various regression and classification problems across different domains.