Dr. Christian Tsoungui Obama

ML Cheat sheet #1

Your practical 7-step guide from business idea to a deployable regression model

A Foundational Guide for Aspiring Data Scientists

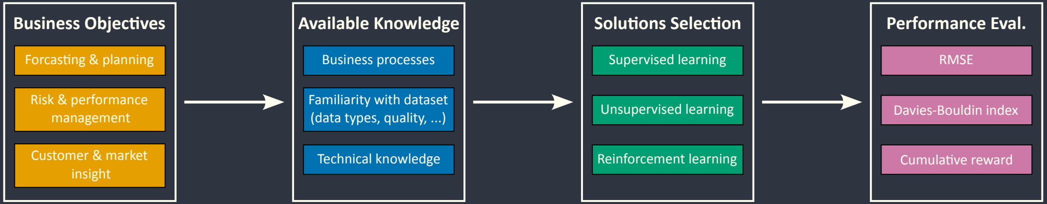
Based on
"Hands-On Machine Learning with Scikit-learn, Keras & TensorFlow"
by

Aurélien Géron

Step 1: Properly Frame the Problem

- Imagine you've been collecting weather data with a small sensor. You wonder: "If I know the humidity, can I predict the temperature?"
- Now think of a store tracking its advertising budget and sales. Someone asks: "If we change our ad budget, can we predict our sales?"

Building machine learning (ML) models for such questions requires to properly frame the problem and understand its context.



Machine learning project roadmap: illustration of key points to consider when framing the problem

Practical steps to build regression models are described in what follows.

Step 2 : Acquire the data

Possible data sources:

- first-party data (e.g., company databases, operational data)
- online sources (e.g., web scraping, APIs)
- public, open data repository (e.g., Kaggle, UCI)

```python
import kagglehub
from kagglehub import KaggleDatasetAdapter

# Loading a DataFrame from Kaggle

adsSales = kagglehub.dataset_load(
                              KaggleDatasetAdapter.PANDAS,
                              "thorgodofthunder/tvradionewspaperadvertising/versions/1",
                              "Advertising.csv",
                                )
```

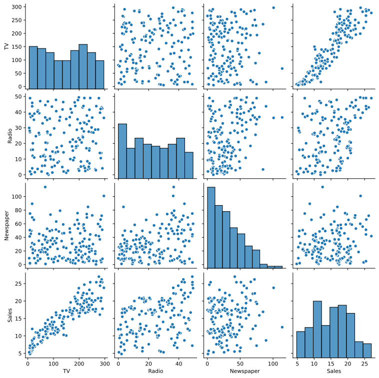
Example code to import "Advertising" dataset from Kaggle[1]

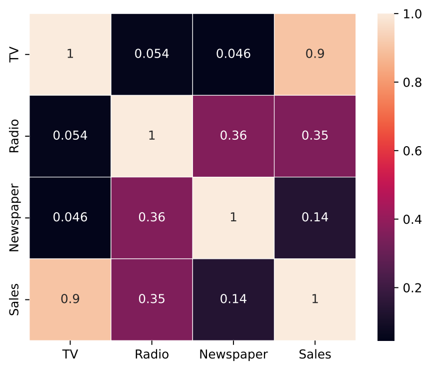
[1] www.kaggle.com/datasets/thorgodofthunder/tvradionewspaperadvertising

Step 3: Explore the data

Exploratory data analysis (EDA)

1. split data into train & test sets
2. investigate linear correlation (numeric attributes)
3. transform data using pipelines (e.g., imputation, one-hot-encoding)





Scatter plot matrix: point cloud resembles a straight line for linearly correlated attributes

Correlation matrix: values quantify strength and direction of linear correlation

Step 4: Train & evaluate candidate models

Some candidate models for regression:

Models	Advantages	Disadvantages
Linear regression	simple, interpretable, fast to train	assumes linearity, sensitive to outliers & multicollinearity
Decision tree regressor	models non-linearity, easy to visualize & interpret	prone to overfitting, sensitive to data change & skewness
Support vector regression (SVR)	models non-linearity, effective for high-dimensional & medium-sized data	computationally expensive for large datasets, difficult to interpret & sensitive to kernel choice

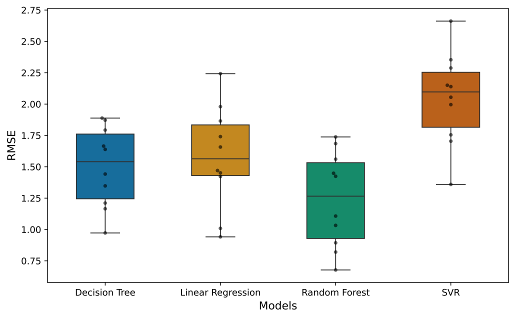
```python
# Training a linear regression model

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

lin_reg = LinearRegression()
lin_reg.fit(adsSales_prepared, adsSales_labels)

adsSales_predictions = lin_reg.predict(
    adsSales_prepared)
lin_mse = mean_squared_error(adsSales_labels,
    adsSales_predictions)

lin_rmse = np.sqrt(lin_mse)
```



Models performance: root mean squared scores (smaller is better)

Step 5 : Fine tune best candidate model (Random forest)

Model hyperparameters are fine-tuned to improve its performance:

1. explore large space with Random search
2. fine tune search with Grid search
3. use cross-validation in both cases

```python
# Hyper-parameters tuning using grid search
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30, 50], 'max_features': [6, 8, 10]},
    {'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',
                           return_train_score=True)

# Fit model for all combinations of parameters
grid_search.fit(adsSales_prepared, adsSales_labels)

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

Step 6: Assess generalization of fine tuned model on test set

Prepare test set with same pipeline as the training set

```
# Defining test dataset
X_test = strat_test_set.drop("Sales", axis=1)
y_test = strat_test_set["Sales"].copy()

X_test_prepared = full_pipeline.transform(X_test)
```

Evaluate model performance

```
# Running predictions on test set
final_predictions = final_model.predict(X_test_prepared)

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

Step 7: Save model for possible deployment

```python
# Saving the model as joblib file

def save_model(model, name="model", model_path="./"):
    """
    Save the model with a given name to a specific location.

    Parameters
    ----------
    model :
        The model trained.
    name:
        The file name with which to save the model
    model_path:
        The path in which the model is saved
        Example:
            "./models/"

    """
    os.makedirs(model_path, exist_ok=True)
    model_name = name + ".pkl"
    pkl_path = os.path.join(model_path, model_name)
    joblib.dump(model, pkl_path)

MODEL_PATH = os.path.join("models", "adsSales")
save_model(final_model, name="Random_forest_V1.1", model_path=MODEL_PATH)
```

Did you find this useful?

Get complete notebook: » GitHub repository «

Comment below: How do you choose your candidate models for regression?

Follow me: for the next cheat sheet in this series.

Like this post: to help it reach more people.

Let's Connect!

/in/christiantsoungui @TsounguiChris