Master 2 — 2020-21 — Université de Bordeaux

Apprentissage automatique appliqué

TD 1 - First experiments on supervised learning

1 A simple example of linear regression

The aim of this exercise is to investigate whether money makes people happy or not:

- the Better Life Index data from OECD website (`https://homl.info/4`);
- the Gross Domestic Product per capita from IMF website (`https://homl.info/5`).

Use the following code to download the data:

```
import urllib.request
import pandas as pd

import os
datapath = os.path.join("datasets", "lifesat", "")
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
os.makedirs(datapath, exist_ok=True)
for filename in ("oecd_bli_2015.csv", "gdp_per_capita.csv"):
    print("Downloading", filename)
    url = DOWNLOAD_ROOT + "datasets/lifesat/" + filename
    urllib.request.urlretrieve(url, datapath + filename)

def prepare_country_stats(oecd_bli, gdp_per_capita):
    oecd_bli = oecd_bli[oecd_bli["INEQUALITY"]=="TOT"]
    oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator", values="Value")
    gdp_per_capita.rename(columns={"2015": "GDP_per_capita"}, inplace=True)
    gdp_per_capita.set_index("Country", inplace=True)
    full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita,
                          left_index=True, right_index=True)
    full_country_stats.sort_values(by="GDP_per_capita", inplace=True)
    remove_indices = [0, 1, 6, 8, 33, 34, 35]
    keep_indices = list(set(range(36)) - set(remove_indices))
    return full_country_stats[["GDP_per_capita", 'Life_satisfaction']].iloc[keep_indices]


oecd_bli = pd.read_csv(datapath + "oecd_bli_2015.csv", thousands=',')
gdp_per_capita = pd.read_csv(
      datapath + "gdp_per_capita.csv",
      thousands=',',delimiter='\t',
      encoding='latin1', na_values="n/a"
)


data_set = prepare_country_stats( oecd_bli, gdp_per_capita)
```

The original dataset is now stored in the table `data_set`. Try finding a linear relationship between *life satisfaction* and the *gross domestic product per capita*. What are your conclusions?

You can use tools from the following libraries to achieve your objective:

```
from matplotlib import pyplot as plt # Plotting library, displaying results
import numpy as np # Linear algebra library
import sklearn # Machine learning library
import pandas as pd  # Database manipulation library
```

More accurately, you can use the following functions: `np.c_`, `plt.plot`, `plt.scatter`, `data_set.plot` and `sklearn.linear_model.LinearRegression`.

2 An exploratory problem of supervised learning

The problem used for this exercise is based in chapter 2 of [1]. It uses the *California Housing Prices dataset* from the *StatLib* repository. A dataset based on data from the 1990 California census.

The dataset contains metrics such as the population, median income, and median housing price for each block group in California. A block is the smallest geographical data for which sample data are published by the US Census Bureau and has typically a population of 600 to 3,000 people.

You are asked to build a tool that will be able to predict the median household price based on other data on the district.

To solve this problem, we follow the checklist presented in [1], Appendix B, page 755.

This list helps you to make sure you're not forgetting crucial steps, however keep in mind that many of the presented points might not be suited or applicable for your specific case.

A) **Frame the problem and look at the big picture.**

1) *Define the objective in business terms.*
2) *How will your solution be used?*
3) *What are the current solutions/workarounds (if any)?*
4) *How should you frame this problem (supervised/unsupervised, online/offline, etc. . .)?*
5) *How should performance be measured?*
6) *Is the performance measure aligned with the business objective?*
7) *What would be the minimum performance needed to reach the business objective?*
8) *What are comparable problems? Can you reuse experience or tools?*
9) *Is human expertise available?*
10) *How would you solve the problem manually?*
11) *List the assumptions you (or others) have made so far.*
12) *Verify assumptions if possible.*

B) **Get the data.**

Note: automate as much as possible so you can easily get fresh data.

1) *List the data you need and how much you need.*
2) *Find and document where you can get that data.*
3) *Check how much space it will take.*
4) *Check legal obligations, and get authorization if necessary.*
5) *Get access authorizations.*
6) *Create a workspace (with enough storage space).*
7) *Get the data.*

   First download the housing datasets with the help of that function:

```
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):
            os.makedirs(housing_path, exist_ok=True)
            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()

fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH)
```

   Now a csv file (`housing.tgz`) is present in the directory `datasets/housing`.

8) *Convert the data to a format you can easily manipulate (without changing the data itself).*

   You can use the panda library to load the csv file.

```
pandas.read_csv? # This function allows to open csv files
```

9) *Ensure sensitive information is deleted or protected (e.g., anonymized).*
10) *Check the size and type of data (time series, sample, geographical, etc.).*

    Inspect the data and check the format, the integrity and the size of the data. You can use the following functions:

```
PANDA_DATA.info?  # to obtain some information on data
PANDA_DATA['KEYS'].value_counts? # counts the frequency for each values
PANDA_DATA.describe? # compute some statistics on the data
PANDA_DATA.hist? # displays an histogram for each column.
```

11) *Sample a test set, put it aside, and never look at it (no data snooping!).*

    You have to cut in two parts the datasets to obtain a testing dataset and a training dataset. This cutting process should follow some rules:

    - cutting should be random
    - a seed should be used to have the same cutting at each execution
    - the distribution of the most important features should be the same between the dataset, the training set and the testing set. In a perfect world, all the features of all those sets should have the same histogram.

You can use the following functions to make some random cutting:

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

Which category is the most important ? Use that category, to make a split where training sets and testing set have the same histogram. To do that, you can construct a simpler histogram for your feature by using:

```
import numpy as np
new_histogram = pd.cut(
      DATA_SET["CATEGORY"],
      bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
      labels=[1, 2, 3, 4, 5]
)
```

Use that histogram with `StratifiedShuffleSplit` to construct the training and the testing set.

Make sure that both the histogram of the testing set and the histogram of the training set are closed for the chosen category.

C) **Explore the data to gain insights.**

Note: try to get insights from a field expert for these steps.

1) *Create a copy of the data for exploration (sampling it down to a manageable size if necessary).*

   Use panda functionality to make a copy of your training set with the following function:

```
PANDA_DATA.copy?  # copy the dataset
```

2) *Create a Jupyter notebook to keep a record of your data exploration.*

3) *Study each attribute and its characteristics:*

   - *Name*
   - *Type (categorical, int/float, bounded/unbounded, text, structured, etc.)*
   - *Noisiness and type of noise (stochastic, outliers, rounding errors, etc.)*
   - *Usefulness for the task*
   - *Type of distribution (Gaussian, uniform, logarithmic, etc.)*

   Use again the following function, in order to analyze data (instead of checking that data is not corrupted):

```
PANDA_DATA.info?  # to obtain some information on data
PANDA_DATA['KEYS'].value_counts? # count the frequency for each values
PANDA_DATA.describe? # compute some statistics on the data
PANDA_DATA.hist? # display an histogram for each column.
```

4) *For supervised learning tasks, identify the target attribute(s).*

5) *Visualize the data.*

   Plot the data by using those usual functions:

```
plt.plot?
data_set.plot?
plt.scatter?
```

   For example, try to make a 2D drawing containing some points where:

   - each point represents a district

- the colored points represent the median value of the price of a house in the corresponding district
- the size of the point is the size of the district's population

6) *Study the correlations between attributes.*

Compute and analyze the correlation matrix of all the training set. Print the scatter matrix of all the training set and analyze it.

You can use the following pandas functions:

```
from pandas.plotting import scatter_matrix
scatter_matrix ? # Display the scatter matrix

PANDAS_OBJECT.corr?  # compute the correlation matrix
```

7) *Study how you would solve the problem manually.*

8) *Identify the promising transformations you may want to apply.*

Try to make some linear combination of categories. In pandas, you can create some new columns by writing:

```
DATA["NEW_CAT_1"] = DATA["CAT_1"]/DATA["CAT_2"]
DATA["NEW_CAT_2"] = 2 * DATA["CAT_1"] + 3 * DATA["CAT_2"]
DATA["NEW_CAT_3"] = np.log(DATA["CAT_1"])
```

Compute the new correlation matrix to observe some improvement with your transformations.

9) *Identify extra data that would be useful (go back to "Get the Data").*

10) *Document what you have learned.*

D) **Prepare the data to better expose the underlying data patterns to Machine Learning algorithm**

Notes:

- *Work on copies of the data (keep the original datasets intact).*
- *Write functions for all data transformations you apply, for five reasons:*
  - *So you can easily prepare the data the next time you get a fresh datasets*
  - *So you can apply these transformations in future projects*
  - *To clean and prepare the test set*
  - *To clean and prepare new data instances once your solution is live*
  - *To make it easy to treat your preparation choices as hyperparameters*

1) *Separate the predictors and the labels*

In panda, you can drop one or more columns by using the drop function:

```
PANDAS_OBJECT.drop ?
```

2) *Data cleaning:*

- *Fix or remove outliers (optional).*
- *Fill in missing values (e.g., with zero, mean, median...) or drop their rows (or columns).*
  Three options are possible to fill missing values.

i. Drop all incomplete rows (drop all entries having some missing value in one of it's categories).

ii. Drop all the incomplete columns (incomplete categories).

iii. Fill missing value with the average or with the median.

Use the following functions to test the three solutions. What are the advantages and drawbacks of each solution?

```
PANDAS_OBJECT.dropna ?
PANDAS_OBJECT.drop ?
PANDAS_OBJECT["CATEGORY"].median ?
PANDAS_OBJECT["CATEGORY"].fillna ?
```

You can use some automatic filling tools by using the sklearn library. For example, you can fill automatically all missing entries with an imputer:

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
imputer.fit(PANDAS_DATA) # Imputer computes the median of all the columns with the existing values
print( imputer.statistics_ ) # print the computed medians
FILLED_DATA = imputer.transform(PANDAS_DATA) # the imputer fill all the missing values with the medians
```

Be careful, inputers can just fill numeric data. So you need to drop all non numeric categories before using it, or to convert all non numeric categories to a numeric one. In order to do this, you can use

```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
```

3) *Feature selection (optional):*

- *Drop the attributes that provide no useful information for the task.*

4) *Feature engineering, where appropriate:*

- *Discretize continuous features.*
- *Decompose features (e.g., categorical, date/time, etc.).*
- *Add promising transformations of features (e.g.,* $\log(x)$, $\sqrt{(x)}$, $\times 2$, . . . *).*
- *Aggregate features into promising new features.*

Construct your own transformation to add new useful attributes by using the sklearn framework:

```
from sklearn.base import BaseEstimator, TransformerMixin

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self): # no *args or **kargs
        pass
    def fit(self, X, y=None):
        # Exctract some usefull information from predictors X and labels y
        return self # nothing else to do
    def transform(self, X):
        CATEGORY_ID = 2
        NEW_CATEGORY = X[:, CATEGORY1_ID] + np.log( 2 * X[:, CATEGORY2_ID_ix] )
        return np.c_[X, NEW_CATEGORY]

attr_adder = CombinedAttributesAdder()
DATAS_WITH_MORE_ATTRIBUTES = attr_adder.transform(PANDA_DATAS.values)
```

5) *Feature scaling:*

- *Standardize or normalize features.*

Use the sklearn framework, to make some min-max standardisation or Normal standardisation. You can use:

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

To finish the data preparation, you have to automate all the previous transformation in a single process. This will be useful for the machine learning process. Use the Pipeline framework of sklearn by importing it with the following lines:

```
from sklearn.pipeline import Pipeline
```

E) **Explore many different models and shortlist the best ones.**

Notes:

- *If the data is huge, you may want to sample smaller training sets so you can train many different models in a reasonable time (be aware that this penalizes complex models such as large neural nets or Random Forests).*
- *Once again, try to automate these steps as much as possible.*

1) *Train many quick-and-dirty models from different categories (e.g., linear, naiveBayes, SVM, Random Forest, neural net, etc.) using standard parameters.*

   In this part, we will use all the algorithms as black boxes.

   Try to train linear regression, and a decision tree, by using:

   Train the following algorithms on your data:

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
```

   Analyze the results of your chosen algorithms with the help of some known metrics. First, list all well-known metrics. Then discuss which one you want to use.

   For example, a usual metric is the mean squared error:

```
from sklearn.metrics import mean_squared_error
```

2) *Measure and compare their performance.*

   - *For each model, use N-fold cross-validation and compute the mean and standard deviation of the performance measure on the N folds.*

   To measure and compare your chosen algorithms, you need to randomly split your training data to make some cross-validation process. Sklearn have a useful framework to do it:

```
from sklearn.model_selection import cross_val_score
```

   This process computes some scores that help you to analyze the result of your algorithms.

3) *Analyze the most significant variables for each algorithm.*

4) *Analyze the types of errors the models make.*

   - *What data would a human have used to avoid these errors?*

5) *Perform a quick round of feature selection and engineering.*

6) *Perform one or two more quick iterations of the five previous steps.*

7) *Shortlist the top three to five most promising models, preferring models that make different types of errors.*

Optional: If you need to store a trained model for a future use, you can use the library `joblib`:

```
import joblib
joblib.dump( my_model, "my_model.pkl" )
my_model = joblib.load( "my_model.pkl" )
```

F) **Fine-tune your models and combine them into a great solution.**

Notes: You will want to use as much data as possible for this step, especially as you move toward the end of fine-tuning.

1) *Fine-tune the hyperparameters using cross-validation:*

   - *Treat your data transformation choices as hyperparameters, especially when you are not sure about them (e.g., if you're not sure whether to replace missing values with zeros or with the median value, or to just drop the rows).*
   - *Unless there are very few hyperparameter values to explore, prefer random search over grid search. If training is very long, you may prefer a Bayesian optimization approach (e.g., using Gaussian process priors, as described by Jasper Snoek et al.).*

   You have inspected previously different algorithms. All those algorithms have different parameters. Some of them are automatically computed by the algorithm. Others are chosen by the user that test it manually. You need to automate the process, to fix some parameters, computes algorithm parameters and compare the result by some cross-validation process. Sklearn propose some frameworks to do it. There is the Grid Search framework and the randomized search framework:

   ```
   from sklearn.model_selection import GridSearchCV
   from sklearn.model_selection import RandomizedSearchCV
   ```

   Use the Grid Search framework to tune a random forest regressor by using the following parameters grid:

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

   Extract some score to analyze the result of the fine-tuning process.

   Extract, from the Grid Search selection model, the best estimator it founds to use it in the future as you favorite predictor.

   Here are some useful attributes:

   ```
   GRID_SEARCH.cv_result_
   GRID_SEARCH.best_estimator_
   GRID_SEARCH.best_params_
   GRID_SEARCH.best_estimator_.feature_importances_
   ```

   Analyze all the results.

2) *Try Ensemble methods. Combining your best models will often produce better performance than running them individually.*

   You can use some ensemble methods to combine your previous algorithms together in order to have better result. Sklearn furnish a framework to manage some well-known ensemble models. For example, try to use the *random forests regressor*:

```
from sklearn.ensemble import RandomForestRegressor
```

3) *Once you are confident about your final model, measure its performance on the test set to estimate the generalization error.*

   THIS IS THE ULTIMATE STEP. When you pass this step, it is over, you will not be allowed to come back and improve your models and their training.

   Now, take the testing data set you reserved during step B), item 11), and use it with your best estimators to compute the generalization error.

   You can also compute a 95 percent confidence interval to estimate the quality of the generalisation error. To do it, you can use:

```
from scipy import stats
stats.t.interval ?
```

   Now, the training is finished ! Now, you have to analyze the results a make a decision on how you will use your trained model. This is the next step of the CheckList.

G) **Present your solution.**

1) *Document what you have done.*
2) *Create a nice presentation.*
   - *Make sure you highlight the big picture first.*
3) *Explain why your solution achieves the business objective.*
4) *Don't forget to present interesting points you noticed along the way.*
   - *Describe what worked and what did not.*
   - *List your assumptions and your system's limitations.*
5) *Ensure your key findings are communicated through beautiful visualizations or easy-to-remember statements (e.g., "the median income is the number-one predictor of housing prices").*

H) **Launch, monitor, and maintain your system.**

1) *Get your solution ready for production (plug into production data inputs, write unit tests, etc.).*
2) *Write monitoring code to check your system's live performance at regular intervals and trigger alerts when it drops.*
   - *Beware of slow degradation: models tend to "rot" as data evolves.*
   - *Measuring performance may require a human pipeline (e.g., via a crowdsourcing service).*
   - *Also monitor your inputs' quality (e.g., a malfunctioning sensor sending random values, or another team's output becoming stale). This is particularly important for online learning systems.*

3) *Retrain your models on a regular basis on fresh data (automate as much as possible).*

References

[1] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn Keras&TensorFlow*, second edition. O'Reilly Media, 2019