Notes on: "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Alexandre De Spiegeleer

December 1, 2020

1 The machine learning landscape

ML is nothing more than fitting a model to the known data!

1.1 Sturcture of a project

1. study the problem
2. Train the ML algo (with lots of data)
3. Solution
4. Check the solution and better understand the problem

→ try to improve model from the observed possible problems

Examples:

- Detection of signals in images using CNNs
- Classifying articles: NLP

1.2 Types of ML

Different categories of ML algorithms:

Properties	**Descrition**	**Examples**
Supervised	Need laballed training data	kNN, Linear Reg., Logistic Reg., SVM, Decision Tree, Random Forest, NN...
Unsupervised	No labelled data, the algorithm puts similar data together	Clustering: k-means, DBSCAN, HCA. Dimensional reduction: PCA, kernel PCA, LLE, t-SNE. Anomaly detection: One-class SVM, Isolation Forest
Semi-Supervised	Both labelled and not labelled data	Deepd belief network, restricted Boltzman machine
Reinforcment Learning	In the learning process give rewards or penalites depending on the action done	
Batch Learning	Trains using ALL data → If needs to retrain, need to train on all the data again	
Online Learning	Train by mini-batches: Train incrementally and can start again from a previous minibatch run	
Instance Based	On new data, check distance to known data and assign same output	
Model-based	Use the data to make predictions / interpolates to new data	

1.2.1 Unsupervised Learning

Feature Extraction Combines related features into a better one (e.g. using PCA)

Anomaly Detection Find anomalies in dataset(e.g. removing some outliers)

Association rule learning Discover relationships in large datasets

1.2.2 semi-supervised Learning

Often much data but only a few are labelled

→ Often combinaison of supervised and unsupervised algo.

1.2.3 Reinforcement Learning

Trains an agent by giving rewards and penalties to obtained the best policy

1.3 Challenges

1.3.1 data

Lack of data ML requires lots of data (mininmum thousands of examples).
Note that models performances increase with number of data.
→ Choice to work on model or gather more data!

Non representative data Data used for training must include similar data to those for predictions. (Poor extrapolation capacity)

Sampling Bias non representative data because the sampling is flawed.
Bias can appear if the proportion of data in different classes are not representative.

Poor data quality if errors, outliers and noise in data

Irrelevant features There should be enough relevant features and not much crap.

1.3.2 Fitting

Overfitting The model fits too well the training data. Occurs because model too complex compared to data → lose predictability.

Underfitting Model too simple compared to the data to be represented e.g. Linear fit on a non-linear problem

Solutions to overfitting:

- *Regularization*
  → Making the model simpler to avoid by adjusting hyperparameter
- *Hyperparameter*
  → Parameters of the learning algorithm that dictates the amount of regularization.

Solutions to underfitting

- Select a better model
- Have better features
- reduce the constraints on the model (e.g. hyperparameters)

1.4 Evaluating performance

Split the data into training set (80%) and testing set (20%). Evaluation on the test set gives an estimate of the error on unseen data. When training multiple models on the training set and testing them on the test sets, our selection of the "best" model is which model best fits the test data set. → Model cannot necessarily be good on new data. Thus, keep a validation set.

Thus, there are 3 sets of data: Training, Validation and Test

1. Train multiple models with different Hyperparameter on the training set
2. Select the model that performs best on the validation set.
3. Verify that the model is indeed good on the test set
4. Deploy

It is common to split the training set to train several models on sub-sets of the training set and train one final model on the whole training set.

If the validation set is too small → imprecise evaluation of performances. The validation set cannot be too large either (compared to training set).
→ use *cross-validation*: it uses several small validation sets. Each model is evaluated once per validation set. You can then average the results of the model on the different smaller validation sets.

1.5 Data mismatch

If there are two types of data in the training set (e.g. not from the same source). This may cause problem in the predictions. How to know if overfitting or problem with the data?

Hold part of the training set (data from one of the source) and see how the model performs on that. If it performs well → the error comes from the mismatched data If it performs poorly → the error comes from the overfitting

If it is because of the data mismatched, is it possible to preprocess these to be more like the rest of the data?

2 End-to-end machine learning project

Main steps of a machine learning project:

1. Look at the big picture
2. Get the data
3. Discover and visualise the data
4. Prepare the data for machine learning algorithms
5. Select a model and train it
6. Fine-tune the model
7. Present the solution
8. Launch, monitor and maintain the system

2.1 Look at the big picture

Get info on the problem to be solved e.g.

- Objectives?
- input and output of the model? (i.e. features and regression/categories?)
- Overall project's pipeline
- Current status and precision → idea for the aimed accuracy
- Evaluate what the model needs: multiple regression (if multiple features), univariate/-multivariate regression (if one/several quatities to predict), continuous flow of data or not, size of the training set, ...
- Select an error measurement
- Check that the assumptions that have been made are reasonable

2.2 Get the data

For reproducibility, it is best that everything is scripted, from the download of the data to the final product.

Create virtual environment

```
python -m virtualenv .venv
```

Download the data

```
import os
import tarfile
import urllib.request

def fetch_data(url, data_path, data_file):
  os.makedirs(data_path, exist_ok=True)
  tgz_path = os.path.join(data_path, data_file)
```

```
urllib.request.urlretrieve(url, tgz_path)
data_tgz = tarfile.open(tgz_path)
data_tgz.extractall(path=data_path)
data_tgz.close()
```

Load the data

```
import pandas as pd

data = pd.read_csv(data_path)
```

Quick info on the data

```
# pandas functions
data.head()
# Info about attributes and number of entries
data.info()
# Get the attribute_str data
data["attribute_str"]
# Counts occurences of the categories in category_str
data["category_str"].value_counts()
# count, mean, ...
data.describe()
```

Distribution of the features

```
import matplotlib.pyplot as plt

data.hist(bins=50, figsize=(20,15))
plt.show()
```

Now we want to get a test set that we will not look at until we have selected the model and we a ready for release. The test gives an indication for the error the model will have on the new data it has never seen (the actual new data without label).

There are different ways to create the test set. One must be careful in the way the train and test sets are created.

- Indeed, we cannot just use random instances that change everytime we run the code. We must use a way that always uses the same instances, even if new data are added! Note that if no new data are added, the problem is simpler. This can be done by creating a unique idea for each instance and spliting by id.
- If a category is particularly important for the prediction, we need to keep the right propertions of this category in the train set and the test set. This is done using stratified sampling:

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2,
                               random_state=42)
# It will split the data following the proportions of
# the categories in category_str
```

```
# Note that if it is not a category but a continuous value,
# one can create a new features which is a bined version
# of the continuous data
for train_index, test_index in split.split(data,
                                    data["category_str"]):
  strat_train_set = data.loc[train_index]
  strat_test_set = data.loc[test_index]

data = strat_train_set
```

Now the proportions are correct. If a category was created, delete it.

2.3 Visualise the data

It is now time to visualise the data and better understand them.

There are many plots and ways to investigate the data, here a few examples:

Scatter Plot

```
data.plot(kind="scatter", x="cat_1", y="cat_2", alpha=alpha,
      s=cat_3, label=label, c=cat_4, cmap=plt.get_cmap("jet"),
      colorbar=True)
      plt.legend()
```

Correlation Correlation between pairs of attributes

```
corr_matrix = data.corr()
corr_matrix["label_var"].sort_values(ascending=False)
```

Scattered matrix plot Scatter of each attributes

```
from pandas.plotting import scatter_matrix

# Possibly too many attributes -> reduce
attributes = [cat1, cat2, cat3]
scatter_matrix(data[attributes], figsize=(12, 8))
```

Combining attributes Sometimes a combination of attributes is better than the attributes separately.

```
data["new_var"] = data["var_1"] / data["var_2"]
# And check the new correlation and hope it's better
corr_matrix = data.corr()
corr_matrix["label_var"].sort_values(ascending=False)
```

2.4 Preparing the data for machine learning

Create functions to automatise the treatment of the data. Preferably in a way that it is general and can be re-used later on.

Data and labels

Start by separating the data and the labels for the train set.

```
data = strat_train_set.drop("label_to_predict", axis=1)
data_labels = strat_train_set["label_to_predict"].copy()
```

Missing Values

If there are missing values, there are different methods

```
# Gets rid of the entities which lack the value in
# cat_to_drop
data.dropna(subset=["cat_to_drop"])
# Get rid of the whole attribute
data.drop("cat_to_drop", axis=1)
# Replace by median value in the whole dataset
median = data["cat_to_fill"].median()
data["cat_to_fill"].fillna(median, inplace=True)
```

This can also be done using sklearn toolbox

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
# need to remove categorical attributes
data_numerical = data.drop("categorical_att", axis=1)
imputer.fit(data_numerical)
# Look at the medians:
imputer.statistics_
# Create a numpy array of transformed data with
# filled values
X = imputer.transform(data_numerical)
data_treated = pd.DataFrame(X,
        columns=data_numerical.columns,
        index=data.data_numerical.index)
```

Categorical Attributes

- replace them by integers from 0 to number of categories-1 and use that for training.

  ```
  from sklearn.preprocessing import OrdinalEncoder
  ordinal_encoder = OrdinalEncoder()
  data_cat = data[["categorical_attribute"]]
  data_cat_encoded = ordinal_encoder.fit_transform(data_cat)
  ```

  There is a problem! The numbers have a relation between each other (bigger/smaller) and that property is not necessarily there in the categorical attributes.

- Instead create a new *onehot* attribute for each of the original categories in the categorical_attribute.

  ```
  from sklearn.preprocessing import OneHotEncoder
  cat_encoder = OneHotEncoder()
  # This will create the new attributes.
  # There will be as many new attributes as there were
  # categories in categorical_attribute.
  data_cat_1hot = cat_encoder.fit_transform(data_cat)
  ```

Custom Transformers

Create own transformers that have *fit* and *fit_transform*. This is usefull when creating a pipeline. It can be done by create a new class.
If it inherits from TransformerMixin, *fit_transform()* gets created automatically. If it also inherits BaseEstimator, there are two more methods: *get_params()* and *set_params()*

```
from sklearn.base import BaseEstimator, TransformerMixin
class CombinedAttributesAdder(BaseEstimator, TransformMixin):
def fit(self, X, y=None):
  # Fit the data X i.e. get the values out of the data
  # and save what must be saved
  return self
  def transform(self, X, y=None):
  # apply to the data the fit by using the saved values
  return

  # Which can be used:
  attr_adder = CombinedAttributesAdder()
  data_extra_attribs = attr_adder.fit_transform(data.values)
```

Feature Scaling

The range of values between the fields can vary a lot.
The algorithm learns better with same range of values for each attribute There are two typical metods: min-max scaling and standardization

```
from sklearn.preprocessing import MinMaxScaler,
                                  StandardScaler

scaler = MinMaxScaler()   # or StandardScaler()
data_scaled = scaler.fit_transform(data)
```

Transformation pipeline

To simplify the consecutive transformation of the data.
Numerical pipeline:

```
from sklearn.pipeline import Pipeline
num_pipeline = Pipeline([
            ('imputer', SimpleImputer(strategy="median")),
            ('attribs_adder', CombinedAttributesAdder()),
            ('std_scaler', StandardScaler())
            ])
data_tr = num_pipeline.fit_transform(data)
```

If numerical and categorical attributes:

```
from sklearn.compose import ColumnTransformer
num_attribs = list(data_numerical)
cat_attribs = ["cat_attributes"]
full_pipeline = ColumnTransformer([
      ("num", num_pipeline, num_attribs),
      ("cat", OneHotEncoder(), cat_attribs)
```

```
        ])
    data_prepared = full_pipeline.fit_transform(data)
```

2.5 Select and Train a model