Subject : Choosing the right machine learning in data science python

Name: Laaroussi Saadeddine

In data science, after categorizing, cleaning and understanding your data, choosing the right machine learning for your data to solve a specific problem is often the next step and this can be quite tedious.

Nowadays, many machine learning algorithms give great results. Therefore, for a specific problem many solutions exist, but which one is the best.

There are two ways to assess a model: the first is by using the data to check if the model is giving the right predictions and the second is to use metrics to ascertain the performance of the model.

Aside from the performance of the model, complexity for the time cost, inference time, training time and cost, dimensionality of data, size of data etc… are all important factors to consider when choosing the model for a specific machine learning problem. Some models might need more time to give optimal predictions while other models need less time but still give good enough results.

For the sake of examples, the flight prediction dataset is going to be used, link: https://www.kaggle.com/datasets/shubhambathwal/flight-price-prediction

Table of contents

Key factors to the performance of a model

The key factors to the performance of a model are:

- The accuracy of the model: how accurate ate the predictions made by the model.
- The interpretability of the model: how easy it is to interpret and understand the results of the model.
- The complexity of the model: how much time does it take to build, train and test the model.
- Inference time: how long does the model take to make predictions.
- The dataset size: how much data is available to use as a training and is the data balanced.
- The dimensionality of data: how many features the model has. Models with a high number of feature tend to be more precise however, this increases cost time and a higher number of data is often needed.
- Does the model meet the business goal?

Steps to evaluate a model

Initialization

After the data set has been cleaned, (data was analyzed, processed and transformed), the ***train_test_split*** function is imported from the ***sklearn.model_selection*** library.

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

This function will split the data into a training set and a test set with ratio decided by the user usually a test size of 0.2 is used.

```
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.2,random_state = 20)
```

Choosing a model

Depending on the task, an appropriate model is chosen. If the output of the data is a number then the problem is a regression problem, otherwise if the output is a class then it is a classification problem. Furthermore, if the input data is labeled then the problem is a supervised learning problem otherwise unsupervised method are used.

Each specific problem has their own methods:

- For regression problems, most known methods are linear regression and logistic regression, both methods can be imported from sklearn.linear_model:
  - from sklearn.linear_model import LinearRegression
  - from sklearn.linear_model import LogisticRegression

- For classification problems, most known methods are K means, KNN (K nearest neighbor), SVM (Support Vector Machine) , Random forest:
    - from sklearn.cluster import KMeans
    - from sklearn.neighbors import KNeighborsClassifier
    - from sklearn.svm import SVC
    - from sklearn.ensemble import RandomForestClassifier
- Some methods can be used for regression and classification problems such as decision trees or neural networks:
    - from sklearn.tree import DecisionTreeRegressor
    - from sklearn.neural_network import MLPClassifier

Performance evaluation

In order to evaluate a model depending on the problem different metrics are used

Classification problem

For classification problems the accuracy is obtained with a confusion matrix. In this case, the confusion matrix shows the number of correct and incorrect predictions made by the classification model compared to the actual outcomes (target value) in the data. The matrix is NxN, where N is the number of target values (classes). Performance of such models is commonly evaluated using the data in the matrix.

The function is the sklearn.metrics library : from sklearn.metrics import confusion_matrix

The accuracy is obtained by summing the common positive target and model values and negative target and model values and dividing them by the number of all predictions. The closer the value is to 1, the more accurate is the model.

F1 score also known as F score or F measure can also be obtained from the recall and precision from the confusion matrix, the library sklearn.metrics contains this measure:

from sklearn.metrics import f1_score

Regression problem

When it comes to regression models different metrics can be used such as:

- Root Mean Squared Error (RMSE): is a popular formula to measure the error rate of a regression model. However, it can only be compared between models whose errors are measured in the same units.
    - from sklearn.metrics import mean_squared_error
- Relative Squared Error: Unlike RMSE, the relative squared error (RSE) can be compared between models whose errors are measured in the different units.
- Mean Absolute Error: The mean absolute error (MAE) has the same unit as the original data, and it can only be compared between models whose errors are measured in the same units. It is usually similar in magnitude to RMSE, but slightly smaller.
    - from sklearn.metrics import mean_absolute_error

- Relative Absolute Error: Like RSE , the relative absolute error (RAE) can be compared between models whose errors are measured in the different units.
- Coefficient of Determination: (R2) summarizes the explanatory power of the regression model and is computed from the sums-of-squares terms.
    - from sklearn.metrics import r2_score

Time cost

Small models

For small models there is no need to predict the time cost since this can be done relatively quick. In fact, time cost can be computed using the timeit library

- import timeit

By calling the function before and after the usage of the model, it is possible to obtain the time taken for training and the inference time for predictions

- start = timeit.default_timer()
- # model traininf or prediction
- stop = timeit.default_timer()
- print('Time: ', stop - start)

Big models

For big models of data it is necessary to predict the time needed for training since these can take several hours and days. There are different deep learning models that predict the time it takes to train a machine learning model.

Some companies offer a cost to train models.

References:

1) https://towardsdatascience.com/considerations-when-choosing-a-machine-learning-model-aa31f52c27f3
2) https://towardsdatascience.com/do-you-know-how-to-choose-the-right-machine-learning-algorithm-among-7-different-types-295d0b0c7f60
3) https://www.saedsayad.com/model_evaluation.htm