Machine Learning

Lecture 07: Libraries

Simone Anotonelli, Soroush H. Zargarbashi, Prof. Dr. Aleksandar Bojchevski

08.07.23

Outline

What we learned so far?

We have learned:

Machine learning problems: Classification, regression, etc.

ML algorithms: Linear regression, decision trees, k-NN.

Evaluation metrics: Accuracy, precision, recall, F1-score, etc.

Model selection methods: Cross-validation, hyperparameter search, etc.

Question is how to use these tools to solve real-world problems?

Now in real world...

What does a data scientist do in real-world? Given a dataset, how does the data scientist extract knowledge or make predictions?

Data preprocessing Cleaning, normalization, feature selection, etc.

Model selection Choose the best model for the problem.

Model evaluation Evaluate the model on unseen data.

For that, we need to know:

- how to visualize the data, and design the procedure based on the insights.
- how to use tools, softwares, and libraries.

Outline

A hands-on approach

Our example dataset: Breast cancer dataset Wisconsin (Diagnostic) dataset.

569 instances, classes are `Benign`, and `Malignant`; 30 features including

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter^2 / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

Loading the data, and primary analysis

Tools for loading and handling data:

- `pandas` for data manipulation.
  With `pandas` you can load the dataset into an object called `DataFrame` – a tabular presentation of the data. You can perform operations like filtering, grouping, etc.
- `numpy` for numerical operations; e.g. matrix multiplication, eigenvalue decomposition, etc.

▷ See "Loading the data" section in the notebook.

Visualizing the data

Tools for visualization:

- `matplotlib` for plotting.
  `matplotlib` includes functions to handle plot canvas (making subplots, saving plots, etc), and functions to draw basic plots (lineplot, barplot, etc).
- `seaborn` for statistical data visualization.
  With `seaborn` you can plot more complex plots like pair plots, heatmaps, etc.

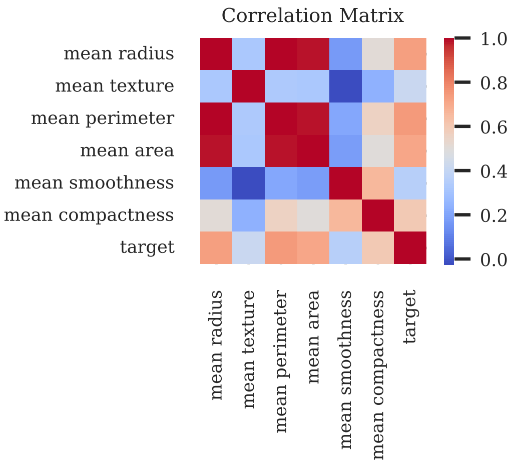
Also for production purposes, you can use `plotly` for interactive plots, or D3JS for more complex and interactive visualization.

Visualization methods

Correlation (heatmap) plot

Represents how two columns are correlated.

- High correlation (close to 1 or -1) between two features can be a flag for redundancy.
- Low correlation (close to 0) between a feature and the target can be a flag for uselessness.



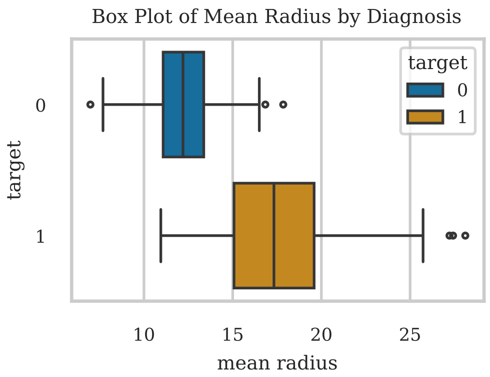
Visualization methods

Box plot

Basic statistics about the feature.

The box ranges between the first to the third quartile, and the line inside the box is the median. The whiskers show the range of the data.

Outliers are shown as dots. Any point that is further than $1.5\times$ either first or third quartile is shown as outlier.

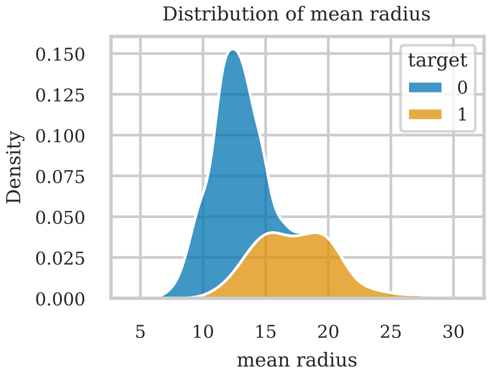


Visualization methods

Kernel density estimate plot

At the center of each datapoint, a kernel function (e.g. Gaussian) is placed. The (normalized) sum of all kernels is the KDE plot.

A bandwidth parameter can be used to control the smoothness of the plot.



Basic visualizations and feature selection

We use visualization to gain information about hidden patterns in the data. For example, we can use box plots to see how the features are distributed in the classes.

▷ See ”Visualizing the data” section in the notebook.

[1]There are more advanced methods like PCA, t-SNE, etc.

Basic visualizations and feature selection

We use visualization to gain information about hidden patterns in the data. For example, we can use box plots to see how the features are distributed in the classes.

▷ See ”Visualizing the data” section in the notebook.

We can use various methods of reduce the dimensionality of the data. High dimensional data can make the ML algorithm unable to learn the patterns.

A simple method to reduce the dimensionality is to remove the features that are not informative. We can use the correlation matrix to see which features are less correlated with the target. We use `DataFrame.corr` in `pandas` for the correlation matrix. [1]

▷ See ”Selecting the features” section in the notebook.

[1]There are more advanced methods like PCA, t-SNE, etc.

Scikit-Learn: transformations

There are some transformations, e.g. preprocessing, feature-selection, etc. that are not learning algorithms. These are also classes that inherit from `BaseEstimator`.

- Similarly, transformations have a `fit` method. They compute their parameters in this function given the training data.
- Instead of `predict` these methods have a `transform` method.

Example: `MinMaxScaler`. In the fit function we compute the minimum and maximum of the training data; $\boldsymbol{x}_{\text{min}}, \boldsymbol{x}_{\text{max}}$.

In the transform function we scale the given data to the range $[0, 1]$ via the formula:

$$\boldsymbol{x}' = \frac{\boldsymbol{x} - \boldsymbol{x}_{\text{min}}}{\boldsymbol{x}_{\text{max}} - \boldsymbol{x}_{\text{min}}}$$

Scikit-Learn: algorithms

Scikit-Learn is a Python library for machine learning. It provides simple and efficient tools for data mining and data analysis. It is built on NumPy, SciPy, and matplotlib.

The logic behind `sklearn` is:

- Each learning algorithm is a class. They all inherit from `BaseEstimator`.
- Each algorithm has a `fit` method to train the model, and a `predict` method to make predictions.
    - `fit` method compute parameters of the model given the training data.
    - `predict` method makes predictions on the test data with the parameters learned in the training phase. [2]

[2] These two methods are sequentially concatenated in the `fit_predict` method.

Spliting the data

Train-test split is a method to evaluate the model on unseen data. The idea is to split the data into two parts: training and test data. The model is selected, and trained on the training data, and the final evaluation is made on test data.

Important note: The preprocessing steps should be applied to the training and test data separately. Any transformation should be applied to the test data using the parameters computed on the training data (avoid data leakage).

Preprocessing

Preprocessing is the process of preparing the data for the learning algorithm. It includes:

- Cleaning the data (removing missing values, etc.)
- Normalizing the data (scaling, etc.)
- Encoding the categorical data (one-hot encoding, etc.)

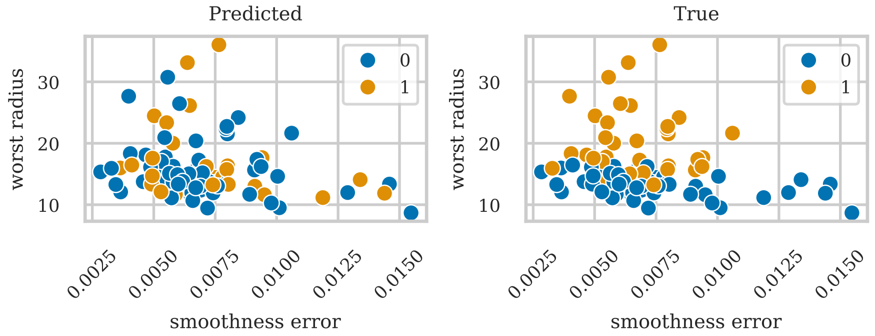
Encoding

Categorical data should be encoded to numerical data. Why? Because most of the learning algorithms are based on numerical computations. There are two main methods for encoding:

- Label encoding: Assigns a unique number to each category.
- One-hot encoding: Creates a binary vector for each category.

Preprocessing: normalization

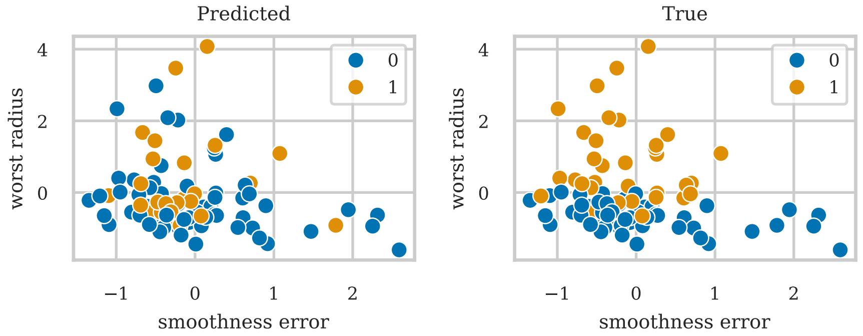
Each feature can have different scales. This can cause problems for the learning algorithm; e.g. k-NN algorithm can be affected by this problem since the distance becomes biased towards the feature with a larger scale.



▷ See "Preprocssing" section in the notebook.

Preprocessing: normalization

Each feature can have different scales. This can cause problems for the learning algorithm; e.g. k-NN algorithm can be affected by this problem since the distance becomes biased towards the feature with a larger scale.



▷ See "Preprocssing" section in the notebook.

Encapsulating the pipeline

The ML procedure at the end can be encapsulated in a pipeline.

The pipeline is a sequence of transformations and a final estimator. The transformations are applied to the data, and the final estimator is trained on the transformed data.

▷ See ”Building the pipeline” section in the notebook.

Why pipeline is useful?

- It wraps around the whole procedure and can be used as a single object (development, and production purposes).
- It can be used in cross-validation, and hyperparameter search.

Hyperparameter search

During data analysis and model selection there are hyperparameters that we should find the best values for; the number of neighbors in k-NN, the depth of the tree in decision trees, the number of top features to select, etc.

Hyperparameter search is the process of finding the best hyperparameters for the model. There are two main methods for hyperparameter search:

- Grid search: It searches all possible combinations of hyperparameters.
- Random search: It searches randomly in the hyperparameter space.

▷ See ”Hyperparameter search” section in the notebook.

Hyperparameter search

Each `Estimator` in `sklearn` has a `get_params` method that returns the hyperparameters of the model.

Similarly, a pipeline has a `get_params` method that returns the hyperparameters of the pipeline.

In the definition of a pipeline, each estimator is assigned with a name; e.g. `('scaler', MinMaxScaler())`. The hyperparameters of the estimator can be accessed via the name; e.g. `pipeline.get_params()['scaler__feature_range']`.

Hyperparameter search

To apply an automated search in the parameter space of a pipeline (or any estimator), we define the range of the hyperparameters in a dictionary. The keys of the dictionary are the names of the hyperparameters, and the values are the range of the hyperparameters.

▷ See ”Hyperparameter search” section in the notebook.

Summary

While in production (or research), we need to use state-of-the-art tools and libraries. Current tools in a data-science and machine-learning pipeline are:

`pandas` for data manipulation, `numpy` for numerical operations, `matplotlib` for plotting, `seaborn` for statistical data visualization, `sklearn` for machine learning.

The `sklearn` library works on a `fit-predict/transform` logic. It provides tools for preprocessing, model selection, evaluation, etc.

The pipeline is a sequence of transformations and estimators. It can be used as a single object for development and production purposes.

Hyperparameter search classes in `sklearn` helps to find the best parameters of any estimator or pipeline.