4.2. Permutation feature importance

Permutation feature importance is a model inspection technique that can be used for any fitted estimator when the data is tabular. This is especially useful for non-linear or opaque estimators. The permutation feature importance is defined to be the decrease in a model score when a single feature value is randomly shuffled [1]. This procedure breaks the relationship between the feature and the target, thus the drop in the model score is indicative of how much the model depends on the feature. This technique benefits from being model agnostic and can be calculated many times with different permutations of the feature.

Warning: Features that are deemed of **low importance for a bad model** (low cross-validation score) could be **very important for a good model**. Therefore it is always important to evaluate the predictive power of a model using a held-out set (or better with cross-validation) prior to computing importances. Permutation importance does not reflect to the intrinsic predictive value of a feature by itself but **how important this feature is for a particular model**.

The `permutation_importance` function calculates the feature importance of estimators for a given dataset. The `n_repeats` parameter sets the number of times a feature is randomly shuffled and returns a sample of feature importances.

Let's consider the following trained regression model:

```
>>> from sklearn.datasets import load_diabetes
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.linear_model import Ridge
>>> diabetes = load_diabetes()
>>> X_train, X_val, y_train, y_val = train_test_split(
...     diabetes.data, diabetes.target, random_state=0)
...
>>> model = Ridge(alpha=1e-2).fit(X_train, y_train)
>>> model.score(X_val, y_val)
0.356...
```

Its validation performance, measured via the R^2 score, is significantly larger than the chance level. This makes it possible to use the `permutation_importance` function to probe which features are most predictive:

```
>>> from sklearn.inspection import permutation_importance
>>> r = permutation_importance(model, X_val, y_val,
...                            n_repeats=30,
...                            random_state=0)
...
>>> for i in r.importances_mean.argsort()[::-1]:
...     if r.importances_mean[i] - 2 * r.importances_std[i] > 0:
...         print(f"{diabetes.feature_names[i]:<8}"
...               f"{r.importances_mean[i]:.3f}"
...               f" +/- {r.importances_std[i]:.3f}")
...
s5      0.204 +/- 0.050
bmi     0.176 +/- 0.048
bp      0.088 +/- 0.033
sex     0.056 +/- 0.023
```

Note that the importance values for the top features represent a large fraction of the reference score of 0.356.

Permutation importances can be computed either on the training set or on a held-out testing or validation set. Using a held-out set makes it possible to highlight which features contribute the most to the generalization power of the inspected model. Features that are important on the training set but not on the held-out set might cause the model to overfit.

The permutation feature importance is the decrease in a model score when a single feature value is randomly shuffled. The score function to be used for the computation of importances can be specified with the `scoring` argument, which also accepts multiple scorers. Using multiple scorers is more computationally efficient than sequentially calling `permutation_importance` several times with a different scorer, as it reuses model predictions.

An example of using multiple scorers is shown below, employing a list of metrics, but more input formats are possible, as documented in Using multiple metric evaluation.

```
>>> scoring = ['r2', 'neg_mean_absolute_percentage_error', 'neg_mean_squared_error']
>>> r_multi = permutation_importance(
...     model, X_val, y_val, n_repeats=30, random_state=0, scoring=scoring)
...
>>> for metric in r_multi:
...     print(f"{metric}")
...     r = r_multi[metric]
...     for i in r.importances_mean.argsort()[::-1]:
...         if r.importances_mean[i] - 2 * r.importances_std[i] > 0:
...             print(f"    {diabetes.feature_names[i]:<8}"
...                   f"{r.importances_mean[i]:.3f}"
...                   f" +/- {r.importances_std[i]:.3f}")
...
r2
    s5      0.204 +/- 0.050
    bmi     0.176 +/- 0.048
    bp      0.088 +/- 0.033
    sex     0.056 +/- 0.023
neg_mean_absolute_percentage_error
    s5      0.081 +/- 0.020
    bmi     0.064 +/- 0.015
    bp      0.029 +/- 0.010
neg_mean_squared_error
    s5      1013.866 +/- 246.445
    bmi     872.726 +/- 240.298
    bp      438.663 +/- 163.022
    sex     277.376 +/- 115.123
```

The ranking of the features is approximately the same for different metrics even if the scales of the importance values are very different. However, this is not guaranteed and different metrics might lead to significantly different feature importances, in particular for models trained for imbalanced classification problems, for which the choice of the classification metric can be critical.

4.2.1. Outline of the permutation importance algorithm

- Inputs: fitted predictive model m, tabular dataset (training or validation) D.
- Compute the reference score s of the model m on data D (for instance the accuracy for a classifier or the R^2 for a regressor).
- For each feature j (column of D):
    - For each repetition k in $1, \ldots, K$:
        - Randomly shuffle column j of dataset D to generate a corrupted version of the data named $\tilde{D}_{k,j}$.
        - Compute the score $s_{k,j}$ of model m on corrupted data $\tilde{D}_{k,j}$.
    - Compute importance i_j for feature f_j defined as:

$$i_j = s - \frac{1}{K} \sum_{k=1}^{K} s_{k,j}$$

4.2.2. Relation to impurity-based importance in trees

Tree-based models provide an alternative measure of feature importances based on the mean decrease in impurity (MDI). Impurity is quantified by the splitting criterion of the decision trees (Gini, Log Loss or Mean Squared Error). However, this method can give high importance to features that may not be predictive on unseen data when the model is overfitting. Permutation-based feature importance, on the other hand, avoids this issue, since it can be computed on unseen data.

Furthermore, impurity-based feature importance for trees are **strongly biased** and **favor high cardinality features** (typically numerical features) over low cardinality features such as binary features or categorical variables with a small number of possible categories.

Permutation-based feature importances do not exhibit such a bias. Additionally, the permutation feature importance may be computed performance metric on the model predictions and can be used to analyze any model class (not just tree-based models).

The following example highlights the limitations of impurity-based feature importance in contrast to permutation-based feature importance: Permutation Importance vs Random Forest Feature Importance (MDI).

4.2.3. Misleading values on strongly correlated features

When two features are correlated and one of the features is permuted, the model will still have access to the feature through its correlated feature. This will result in a lower importance value for both features, where they might *actually* be important.

One way to handle this is to cluster features that are correlated and only keep one feature from each cluster. This strategy is explored in the following example: Permutation Importance with Multicollinear or Correlated Features.

Examples:

- Permutation Importance vs Random Forest Feature Importance (MDI)
- Permutation Importance with Multicollinear or Correlated Features

References:

[1]
L. Breiman, "Random Forests", Machine Learning, 45(1), 5-32, 2001.