Permutation Importance

eli5 provides a way to compute feature importances for any black-box estimator by measuring how score decreases when a feature is not available; the method is also known as “permutation importance” or “Mean Decrease Accuracy (MDA)”.

A similar method is described in Breiman, “Random Forests”, Machine Learning, 45(1), 5-32, 2001 (available online at https://www.stat.berkeley.edu/%7Ebreiman/randomforest2001.pdf).

Algorithm

The idea is the following: feature importance can be measured by looking at how much the score (accuracy, F1, R^2, etc. - any score we’re interested in) decreases when a feature is not available.

To do that one can remove feature from the dataset, re-train the estimator and check the score. But it requires re-training an estimator for each feature, which can be computationally intensive. Also, it shows what may be important within a dataset, not what is important within a concrete trained model.

To avoid re-training the estimator we can remove a feature only from the test part of the dataset, and compute score without using this feature. It doesn’t work as-is, because estimators expect feature to be present. So instead of removing a feature we can replace it with random noise - feature column is still there, but it no longer contains useful information. This method works if noise is drawn from the same distribution as original feature values (as otherwise estimator may fail). The simplest way to get such noise is to shuffle values for a feature, i.e. use other examples’ feature values - this is how permutation importance is computed.

The method is most suitable for computing feature importances when a number of columns (features) is not huge; it can be resource-intensive otherwise.

Model Inspection

For sklearn-compatible estimators eli5 provides `PermutationImportance` wrapper. If you want to use this method for other estimators you can either wrap them in sklearn-compatible objects, or use `eli5.permutation_importance` module which has basic building blocks.

For example, this is how you can check feature importances of sklearn.svm.SVC classifier, which is not supported by eli5 directly when a non-linear kernel is used:

```
import eli5
from eli5.sklearn import PermutationImportance
from sklearn.svm import SVC

# ... load data

svc = SVC().fit(X_train, y_train)
perm = PermutationImportance(svc).fit(X_test, y_test)
eli5.show_weights(perm)
```

If you don't have a separate held-out dataset, you can fit `PermutationImportance` on the same data as used for training; this still allows to inspect the model, but doesn't show which features are important for generalization.

For non-sklearn models you can use `eli5.permutation_importance.get_score_importances()`:

```
import numpy as np
from eli5.permutation_importance import get_score_importances

# ... load data, define score function
def score(X, y):
    y_pred = predict(X)
    return accuracy(y, y_pred)

base_score, score_decreases = get_score_importances(score, X, y)
feature_importances = np.mean(score_decreases, axis=0)
```

Feature Selection

This method can be useful not only for introspection, but also for feature selection - one can compute feature importances using `PermutationImportance`, then drop unimportant features using e.g. sklearn's SelectFromModel or RFE. In this case estimator passed to `PermutationImportance` doesn't have to be fit; feature importances can be computed for several train/test splits and then averaged:

```python
import eli5
from eli5.sklearn import PermutationImportance
from sklearn.svm import SVC
from sklearn.feature_selection import SelectFromModel

# ... load data

perm = PermutationImportance(SVC(), cv=5)
perm.fit(X, y)

# perm.feature_importances_ attribute is now available, it can be used
# for feature selection - let's e.g. select features which increase
# accuracy by at least 0.05:
sel = SelectFromModel(perm, threshold=0.05, prefit=True)
X_trans = sel.transform(X)

# It is possible to combine SelectFromModel and
# PermutationImportance directly, without fitting
# PermutationImportance first:
sel = SelectFromModel(
    PermutationImportance(SVC(), cv=5),
    threshold=0.05,
).fit(X, y)
X_trans = sel.transform(X)
```

See `PermutationImportance` docs for more.

Note that permutation importance should be used for feature selection with care (like many other feature importance measures). For example, if several features are correlated, and the estimator uses them all equally, permutation importance can be low for all of these features: dropping one of the features may not affect the result, as estimator still has an access to the same information from other features. So if features are dropped based on importance threshold, such correlated features could be dropped all at the same time, regardless of their usefulness. RFE and alike methods (as opposed to single-stage feature selection) can help with this problem to an extent.