Data Mining und Maschinelles Lernen

Prof. Kristian Kersting
Steven Lang
Felix Friedrich

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 2022
Übungsblatt 5

Benötigte Dateien
Alle benötigten Datensätze und Skriptvorlagen finden Sie in unserem Moodle-Kurs:
https://moodle.informatik.tu-darmstadt.de/course/view.php?id=1058

5.1 Bagging - Zufallswälder

Ein Zufallswald (engl. Random Forrest) ist ein Meta-Schätzer, der eine Reihe von Entscheidungsbaum-Klassifikatoren auf verschiedene Unterstichproben des Datensatzes anpasst und die Mittelwertbildung verwendet, um die Vorhersagegenauigkeit zu verbessern und Überanpassung (engl. Overfitting) zu kontrollieren. Dies wird als Bagging-Methode bezeichnet. Bei Ensemble-Algorithmen bilden Bagging-Methoden eine Klasse von Algorithmen, die mehrere Instanzen eines Black-Box-Schätzers auf zufälligen Untermengen des ursprünglichen Trainingssatzes aufbauen und dann ihre individuellen Vorhersagen zu einer endgültigen Vorhersage aggregieren. Diese Methoden werden als eine Möglichkeit verwendet, die Varianz eines Basisschätzers (z.B. eines Entscheidungsbaums) zu reduzieren, indem man die Randomisierung in sein Konstruktionsverfahren einführt und dann ein Ensemble daraus macht. In vielen Fällen stellen Bagging-Methoden eine sehr einfache Möglichkeit zur Verbesserung gegenüber einem einzelnen Modell dar, ohne dass der zugrundeliegende Basisalgorithmus angepasst werden muss. Da sie eine Möglichkeit zur Verringerung der Überanpassung bieten, funktionieren Bagging-Methoden am besten mit starken und komplexen Modellen (z.B. tiefen Entscheidungsbäumen), im Gegensatz zu Boosting-Methoden, die normalerweise am besten mit schwachen Modellen (z.B. flachen Entscheidungsbäumen) funktionieren.

a) **Entscheidungsbaum vs. Zufallswald (diskret)**

Wir möchten in einer einfachen Demonstration die Perfomanz von Entscheidungsbäumen mit Zufallswäldern vergleichen. Wir werden beide Modelle anhand eines Satzes von Klassifizierungs- und Regressionsdatensätzen kreuzvalidieren.

Implementieren Sie die Funktion `cross_val_dt_rt`, welche eine 10-fache Kreuzvalidierung für einen `DecisionTreeClassifier` mit Standardparametern und `RandomForestClassifier` mit 20 Schätzern durchführt. Sie können dabei die Funktion `sklearn.model_selection.cross_validate` verwenden. Abschließend sollen die Genauigkeitsscores zurückgegeben werden.

```python
def cross_val_dt_rt(load_fun) -> (dict, dict):
    """Conducts a 10-fold cross validation for a decision tree and random forrest and
     returns their accuracy scores given an sklearn discrete dataset loader function."""
    # Load data
    X, y = load_fun(return_X_y=True)
    # Cross-validate Decision Tree and Random Forest
    scores_dt = cross_validate(
        estimator=DecisionTreeClassifier(),
        X=X,
        y=y,
        scoring="accuracy",
        cv=10,
        return_train_score=True,
        n_jobs=-1,
    )
    scores_rf = cross_validate(
        estimator=RandomForestClassifier(n_estimators=20),
        X=X,
        y=y,
        scoring="accuracy",
```

```
        cv=10,
        return_train_score=True,
        n_jobs=-1,
    )
    return scores_dt, scores_rf
```

b) **Entscheidungsbaum vs. Zufallswald (kontinuierlich)**

Es wird deutlich, dass ein Random Forest einen Entscheidungsbaum in Testvorhersagegenauigkeit übertrifft. Wir können dies für Datensätze mit kontinuierlichen Zielvariablen wiederholen.

Führen Sie eine 10-fache Kreuzvalidierung in der Funktion `cross_val_dt_rf_continuous` für kontinuierliche Zielvariablen durch. Es sollen hierbei erneut 20 Schätzer verwendet werden und die mittlere quadratische Abweichung als Metrikscore zurückgegeben werden.

```
def cross_val_dt_rf_continuous(load_fun) -> (dict, dict):
    """Conducts a 10-fold cross validation for a decision tree and random forrest and
     returns their mse scores given an sklearn discrete dataset loader function."""
    # Load data
    X, y = load_fun(return_X_y=True)
    # Cross-validate Decision Tree and Random Forest
    scores_dt = cross_validate(
        estimator=DecisionTreeRegressor(),
        X=X,
        y=y,
        scoring=make_scorer(mean_squared_error),
        cv=10,
        return_train_score=True,
        n_jobs=-1,
    )
    scores_rf = cross_validate(
        estimator=RandomForestRegressor(n_estimators=20),
        X=X,
        y=y,
        scoring=make_scorer(mean_squared_error),
        cv=10,
        return_train_score=True,
        n_jobs=-1,
    )
    return scores_dt, scores_rf
```

5.2 Zufallswälder - Hyperparameter

Was wir beobachten, ist, dass der Random Forest nicht nur den Entscheidungsbaum übertrifft, sondern auch die Trainingsdaten nicht so sehr überanpasst, wie der Entscheidungsbaum. Für jeden Datensatz passt sich der einzelne Baum vollständig an die Trainingsdaten an und erreicht einen mittleren quadratischen Wurzelfehler von 0.0, während der Zufallswald nicht vollständig mit den Trainingsdaten übereinstimmt. Auf der anderen Seite erzielt der Zufallswald in jedem Fall ein besseres Testergebnis. Dies deutet darauf hin, dass Zufallswälder bessere Verallgemeinerungsfähigkeiten haben als Entscheidungsbäume.

a) **Anzahl der Bäume**

Der wichtigste Parameter eines Zufallswaldes ist seine Ensemblegröße, d.h. wie viele Entscheidungsbäume verwendet werden, um eine Mehrheitsabstimmung für die Entscheidung durchzuführen. Beginnen wir mit dem Zifferndatensatz und zeigen wir den Einfluss der Ensemblegröße auf die Klassifikationsgenauigkeit.

Um den Einfluss der Anzahl der Bäume zu sehen, werden wir jeden Wert im Intervall $n \in [1, \cdots, 40]$ kreuzvalidieren und ein `RandomForestClassifier`-Modell mit n-Entscheidungsbäumen erstellen. Evaluieren Sie die Anzahl der Schätzer in Funktion `evaluate_n_estimators` mittels einer 10-fachen Kreuzvalidierung und geben sie die mittlere Trainings- und Testgenauigkeit zurück.

```
def evaluate_n_estimators(X: np.ndarray, y: np.ndarray, n: int) -> (float, float):
    """Run 10 fold cross-validation of the model for a given number of trees and returns the
    mean train and test score."""
    model = RandomForestClassifier(n_estimators=n)
```

```
    scores = cross_validate(
        estimator=model, X=X, y=y, scoring="accuracy", cv=10, return_train_score=True, n_jobs=-1,
    )
    return np.mean(scores["train_score"]), np.mean(scores["test_score"])
```

Was können Sie bei zunehmender Anzahl von Bäumen im Ensemble feststellen?

Wir können beobachten, dass mit zunehmender Anzahl von Bäumen im Ensemble auch die Trainings- und Testgenauigkeit zunimmt. Da die Erhöhung der Anzahl der Bäume zu einer Verringerung der Verzerrung und Varianz führt, stellen wir auch fest, dass das Modell mit zunehmender Komplexität nicht an die Trainingsdaten überangepasst wird.

b) **Maximal Baumtiefe**

Ein weiterer interessanter Aspekt, den es zu betrachten gilt, ist die Komplexität der Entscheidungsbäume. Wir können den Einfluss der Komplexität der Lerner auf die Leistung des Ensembles untersuchen, indem wir das obige Experiment wiederholen, aber statt mit einer unterschiedlichen Anzahl von Bäumen zu evaluieren, werden wir diesmal die maximale Baumtiefe für jeden Baum mit einer festen Anzahl von Bäumen ändern.

Führen Sie eine 10-fache Kreuzvalidierung für gegebene Baumtiefe mit 20 Schätzern in der Funktion `evaluate_depth` durch. Was fällt ihnen bei größer werdender Baumtiefe auf?

```
def evaluate_depth(X: np.ndarray, y: np.ndarray, depth: int) -> (float, float):
    """Run 10 fold cross-validation of the model for a given tree depth and returns the
    mean train and test score."""
    model = RandomForestClassifier(max_depth=depth, n_estimators=20)
    scores = cross_validate(
        estimator=model, X=X, y=y, scoring="accuracy", cv=10, return_train_score=True, n_jobs=-1,
    )
    return np.mean(scores["train_score"]), np.mean(scores["test_score"])
```

Es wird deutlich, dass bei Zufallswäldern das Ensemble umso stärker wird, je stärker die Entscheidungsbäume sind, ohne dass die Verallgemeinerung verloren geht. Dies ist eine wichtige Beobachtung, da bei anderen Algorithmen, die auf einem Satz von Basislernern basieren, schwache Lerner bevorzugt werden können, oder leistungsschwache Lerner ebenso wie starke Lerner, wie z.B. im Fall von Boosting.

c) **Anzahl der Merkmale**

Um die Korrelation zu reduzieren und Zufälligkeit in das Ensemble zu induzieren, wählt der Zufallswald-Algorithmus eine zufällige Teilmenge von k-Merkmalen aus allen verfügbaren Eingabemerkmalen für jeden internen Entscheidungsbaum aus. Die Verwendung einer geringeren Anzahl von Eingabemerkmalen verringert die Ähnlichkeit zwischen den einzelnen Bäumen, führt aber auch zu weniger komplexen und daher schwächeren Bäumen. Andererseits wird durch die Erhöhung der Anzahl der Merkmale jeder Baum leistungsstärker, aber auch die Korrelation zwischen den Bäumen erhöht (die bei Verwendung *aller* Merkmale maximiert wird).

Evaluieren Sie den den Einfluss dieses Parameters, indem Sie eine 10-fache Kreuzvalidierung mit 20 Schätzern für alle mögliche Anzahl von Merkmalen im Zifferndatensatz in der Funktion `evaluate_features` durchführen.

```
def evaluate_features(X: np.ndarray, y: np.ndarray, n_features: int) -> (float, float):
    """Run 10 fold cross-validation of the model for a given number of features per tree and returns
     the
    mean train and test score."""
    model = RandomForestClassifier(max_features=n_features, n_estimators=20)
    scores = cross_validate(
        estimator=model, X=X, y=y, scoring="accuracy", cv=10, return_train_score=True, n_jobs=-1,
    )
    return np.mean(scores["train_score"]), np.mean(scores["test_score"])
```

Welche Faustregel wird in der Literatur oft verwendet, um die Anzahl der Merkmale festzulegen? Ist diese Regel auch in diesem Fall sinnvoll?

Die Ergebnisse bestätigen, dass ein Wert um den gebildeten Schätzwert von $k = \sqrt{n\_features} = \sqrt{64} = 8$*, den man in der wissenschaftlichen Literatur bezüglich eines guten Wertes für die Anzahl der Merkmale finden kann, zu einer guten Wahl von* k *führt.*