Data Mining und Maschinelles Lernen

Prof. Kristian Kersting
Steven Lang
Felix Friedrich

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 2022
Übungsblatt 4

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

4.1 Entscheidungsbäume - Vor- und Nachteile

Entscheidungsbäume (engl. Decision Tree (DTs)) sind eine nichtparametrische, überwachte Lernmethode, die zur Klassifizierung und Regression verwendet wird. Ziel ist es, ein Modell zu erstellen, das den Wert einer Zielvariablen vorhersagt, indem einfache Entscheidungsregeln gelernt werden, die aus den Datenmerkmalen abgeleitet werden.

Entscheidungsbäume lernen Wenn-dann-sonst-Entscheidungsregeln um einen Datensatz zu approximieren. Je tiefer der Baum ist, desto komplexer sind die Entscheidungsregeln und desto passender ist das Modell.

a) **Vorteile**

Nennen Sie drei Vorteile von Entscheidungsbäumen:

- Einfach zu verstehen und zu interpretieren. Bäume können visualisiert werden.

- Erfordert wenig Datenaufbereitung. Andere Techniken erfordern oft eine Datennormalisierung, es müssen Dummy-Variablen erstellt und Leerwerte entfernt werden. Beachten Sie jedoch, dass dieses Modell fehlende Werte nicht unterstützt.

- Die Kosten für die Verwendung des Baums (d.h. für die Vorhersage von Daten) sind logarithmisch in der Anzahl der Datenpunkte, die zum Trainieren des Baums verwendet werden.

- Kann sowohl numerische als auch kategorische Daten verarbeiten. Andere Techniken sind in der Regel auf die Analyse von Datensätzen spezialisiert, die nur einen Variablentyp aufweisen.

- Kann Probleme mit mehreren Ausgaben handhaben.

- Verwendet ein White-Box-Modell. Wenn eine gegebene Situation in einem Modell beobachtbar ist, lässt sich die Erklärung für die Bedingung leicht durch boolesche Logik erklären. Im Gegensatz dazu können in einem Black-Box-Modell (z.B. in einem künstlichen neuronalen Netz) die Ergebnisse schwieriger zu interpretieren sein.

- Es ist möglich, ein Modell durch statistische Tests zu validieren. Dadurch ist es möglich, die Zuverlässigkeit des Modells zu berücksichtigen.

- Funktioniert auch dann gut, wenn die Annahmen durch das wahre Modell, aus dem die Daten generiert wurden, in gewisser Weise verletzt werden.

b) **Nachteile**

Nennen Sie drei Nachteile von Entscheidungsbäumen:

- Entscheidungsbaum-Lernalgorithmen können überkomplexe Bäume erstellen, die die Daten nicht gut verallgemeinern. Dies wird als Überanpassung bezeichnet. Um dieses Problem zu vermeiden, sind Mechanismen wie das Beschneiden, das Festlegen der minimalen Anzahl von Stichproben, die an einem Blattknoten erforderlich sind, oder das Festlegen der maximalen Tiefe des Baumes erforderlich.

- Entscheidungsbäume können instabil sein, weil kleine Abweichungen in den Daten dazu führen können, dass ein völlig anderer Baum erzeugt wird. Dieses Problem wird durch die Verwendung von Entscheidungsbäumen innerhalb eines Ensembles gemildert.

- Es ist bekannt, dass das Problem des Lernens eines optimalen Entscheidungsbaums unter verschiedenen Optimalitätsaspekten und selbst bei einfachen Konzepten NP-vollständig ist. Folglich basieren praktische Algorithmen zum Lernen von Entscheidungsbäumen auf heuristischen Algorithmen wie dem Greedy-Algorithmus, bei dem an jedem Knoten lokal optimale Entscheidungen getroffen werden. Solche Algorithmen können nicht garantieren, dass sie den global optimalen Entscheidungsbaum zurückgeben. Dies kann abgeschwächt werden, indem mehrere Bäume in einem Ensemble-Lernalgorithmus trainiert werden, wobei die Merkmale und Stichproben nach dem Zufallsprinzip mit Ersetzung beprobt werden.

- Es gibt Konzepte, die schwer zu erlernen sind, weil Entscheidungsbäume sie nicht leicht ausdrücken, wie z.B. XOR, Paritäts- oder Multiplexerprobleme.

- Entscheidungsbaum-Lernalgorithmen erstellen verzerrte Bäume, wenn einige Klassen dominieren. Es wird daher empfohlen, den Datensatz vor der Anpassung an den Entscheidungsbaum auszugleichen.

4.2 Entscheidungsbäume - Klassifikation

`sklearn.tree.DecisionTreeClassifier` ist eine Klasse, die eine Mehrklassenklassifizierung eines Datensatzes durchführen kann.

Wie bei anderen Klassifikatoren nimmt `DecisionTreeClassifier` zwei Arrays als Eingabe entgegen: ein Array `X`, *sparse* oder *non-sparse*, mit der Größe `[n_samples, n_features]`, das die Trainingsproben enthält, und ein Array `Y` mit ganzzahligen Werten, mit der Größe `[n_samples]`, das die Klassenbezeichnungen für die Trainingsproben enthält.

a) **Unsichere Vorhersagen**

Entscheidungsbäume erlauben es eine (Pseudo)-Wahrscheinlichkeit für die Entscheidungen anzugeben. Finden Sie alle Entscheidungen des Modells für den Testdatensatz, für welches das Modell keine 100 % Zuordnung treffen konnte. Verwenden Sie dabei die Methode `clf.predict_proba`.

```python
def get_unsure_test_items(clf, X_test: np.ndarray) -> np.ndarray:
    """Returns a boolean array where all items with a probability != 1 are marked as true"""
    y_test_prob = clf.predict_proba(X_test)
    return y_test_prob.max(axis=1) != 1
```

b) **Entscheidungsregionen**

Um die Entscheidungsregionen zu visualisieren, können wir eine Vorhersage für jeden möglichen Datenpunkt um die gültige Region um den 2D Iris-Datensatz machen.

Stellen Sie die Entscheidungsregionen In der Methode `plot_decision_boundary` dar.

Unterscheiden Sie dabei visuell die Trainings- und Testmenge und markieren Sie alle Punkte von a), bei welchem das Modell keine klare Entscheidung treffen konnte. Sie können dabei den Quelltext von `plot_decision_boundary` von `dmml_u2.zip/utils.py` als Referenz verwenden und erweitern.

```python
def plot_decision_boundary(clf, X_train: np.ndarray, X_test: np.ndarray,
                           y_train: np.ndarray, y_test: np.ndarray, y_test_unsure: np.ndarray) ->
    None:
    """Create a decision boundary plot that shows the predicted label for each point."""
    cache = {}

    # Plot decision regions
    plt.figure()
    plt.title("Decision Tree Decision Regions on 2D Iris")

    h = 0.05  # step size in the mesh

    # Create color maps
    cmap_light = ListedColormap(["#FFAAAA", "#AAFFAA", "#AAAAFF"])
    cmap_bold = ListedColormap(["#FF0000", "#00FF00", "#0000FF"])

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
    y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    k = clf.min_samples_leaf
    if k in cache:
        Z = cache[k]
    else:
        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
        cache[k] = Z

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot also the training and test points
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap_bold, marker="x", edgecolor="k", s
    =25, label="Training Points")
    plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cmap_bold, marker="^", edgecolor="k", s
    =25, label="Test Points")
```

```
    plt.scatter(X_test[y_test_unsure][:, 0], X_test[y_test_unsure][:, 1], c="yellow", marker="*",
    edgecolor="k", s=50, label="Unsure Points")

    plt.legend()
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.show()
```

c) **Kreuzvalidierung**

Wir können visuell erkennen, dass der Entscheidungsbaum mit zunehmender Tiefe dazu neigt, sich zu spezialisieren, um zu den einzelnen Datenpunkten zu passen, was zu komplexeren Entscheidungsregionen führt. Um den optimalen Wert für die maximale Tiefe zu finden, können wir außerdem eine Kreuzvalidierung des 2D-Iris-Datensatzes über einen Bereich möglicher Werte für den Parameter für die maximale Tiefe durchführen.

Implementieren Sie die Methode `get_acc_per_depth`, welche eine 10-fache Kreuzvalidierung durchführt und die korrespondierenden Train- und Testgenauigkeiten zurückgibt.

```
def get_acc_per_max_depth(X: np.ndarray, y: np.ndarray, max_depths: range) -> [np.ndarray, np.ndarray
    ]:
    """Runs 10-fold cross validation for all given depths and
     returns an array for the corresponding train and test accuracy"""
    def evaluate(X, y, max_depth):
        model = tree.DecisionTreeClassifier(max_depth=max_depth)
        scores = cross_validate(
            estimator=model, X=X, y=y, scoring="accuracy", cv=10, return_train_score=True
        )
        return np.mean(scores["train_score"]), np.mean(scores["test_score"])

    # Evaluate each depth parameter
    accuracies = np.array([evaluate(X, y, md) for md in max_depths])
    acc_train = accuracies[:, 0]
    acc_test = accuracies[:, 1]
    return acc_train, acc_test
```

4.3 Entscheidungsbäume - Regression

Entscheidungsbäume können auch zur Regression verwendet werden. Dazu erstellen wir eine verrauschte Sinusfunktion. Wir können jetzt Entscheidungsbäume anpassen, um die Sinusfunktion zu erlernen. Als Ergebnis werden lokale lineare Regressionen gelernt, die sich der Sinuskurve annähern.

a) **Regression**

Verwenden Sie den `sklearn.tree.DecisionTreeRegressor` in der Methode `get_dt_prediction` um eine Regression mittels eines Entscheidungsbaum anzuwenden. Geben Sie anschließend die Vorhersagen des Modells für gegebenen Trainingdatensatz zurück.

```
def get_dt_prediction(X_train: np.ndarray, y_train: np.ndarray, max_depth: int) -> np.ndarray:
    """Fits a decision tree regressor and returns its predictions as an array."""
    # Fit regression model
    regr_1 = DecisionTreeRegressor(max_depth=max_depth)
    regr_1.fit(X_train, y_train)
    # Predict
    y_pred = regr_1.predict(X_train)
    return y_pred
```

b) **Ergebnisvisualisierung**

Wir können sehen, dass, wenn die maximale Tiefe des Baumes (gesteuert durch den Parameter `max_depth`) zu hoch eingestellt ist, die Entscheidungsbäume zu feine Details der Trainingsdaten lernen und aus dem Rauschen lernen, d.h. sie *überanpassen* sich.

Zeigen Sie die verschiedenen Anpassungen für die unterschiedlichen Einstellungen von `max_depth` in der Methode `plot_regression_models` an.

```python
def plot_regression_models(X_cont: np.ndarray, y_cont: np.ndarray, y_1: np.ndarray, y_2: np.ndarray):
    """Plots the regression results for y_1 and y_2 on top of the training data X_cont, y_cont."""
    plt.figure(figsize=(12, 4))
    ax = plt.subplot(121)
    plt.scatter(X_cont, y_cont, s=20, edgecolor="black")
    plt.plot(X_cont, y_1, label="$max\_depth=2$", color="orange")
    ax.legend()
    ax.set_ylabel("Target")
    ax.set_xlabel("Data")
    ax = plt.subplot(122)
    plt.scatter(X_cont, y_cont, s=20, edgecolor="black")
    plt.plot(X_cont, y_2, label="$max\_depth=5$", color="green")
    ax.legend()
    plt.xlabel("Data")
    plt.suptitle("Decision Tree Regression")
    plt.show()
```