ZASTOSOWANIE MODELU KLASYFIKATORA DRZEWA DECYZYJNEGO DO PRZEWIDYWANIA BEZPIECZEŃSTWA SAMOCHODU

Decision Tree Classifier

Paweł Frąckowiak - 47620

(Polecenie: Wykorzystanie systemu Spark i biblioteki MLlib do wykonania obliczeń opartych na uczeniu maszynowym dla dużego zbioru danych)

Informatyka, semestr 6 Inteligentne aplikacje Laboratorium nr 7 i 8

WPROWADZENIE

Decision-Tree Classifier, czyli klasyfikator drzewa decyzyjnego, to model uczenia maszynowego wykorzystywany do rozwiązywania problemów klasyfikacji. Działa na zasadzie tworzenia drzewa decyzyjnego, gdzie cechy samochodu są wykorzystywane do podejmowania decyzji dotyczących jego bezpieczeństwa.

Dzięki zastosowaniu klasyfikatora drzewa decyzyjnego, możemy analizować różne czynniki, takie jak parametry techniczne, wyposażenie czy koszt samochodu, i na ich podstawie przewidzieć, czy samochód jest bezpieczny.

Ten model jest użyteczny zarówno dla indywidualnych użytkowników decydujących się na samochód, jak i dla producentów, którzy mogą wykorzystać wyniki analizy do doskonalenia projektów i wprowadzania ulepszeń mających na celu zwiększenie bezpieczeństwa samochodów.

IMPLEMENTACJA

W związku z obecnością gotowych implementacji algorytmów uczenia maszynowego w bibliotece MLlib, będziemy korzystać właśnie z niej, aby importować odpowiednie moduły i funkcje. Poniżej w krokach został przedstawiony cały proces implementacji:

1. **Otwarcie sesji platformy Spark:**

```
import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
sc = spark.sparkContext
print(spark.version)
```

2. **Dołączenie niezbędnych bibliotek:**

```
import findspark
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
```

3. **Zaimportowanie danych:**

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

struct_car = StructType([ \
StructField("buying", StringType(), True),
StructField("maint", StringType(), True),
StructField("doors", StringType(), True),
StructField("persons", StringType(), True),
StructField("lug_boot", StringType(), True),
StructField("safety", StringType(), True),
StructField("class", StringType(), True)])

df = spark.read.format("csv") \
.option("header", "false") \
.option("delimiter", ",") \
.schema(struct_car) \
.load("car.data")

df.show(10)
```

```
+------+-----+-----+-------+--------+------+-----+
|buying|maint|doors|persons|lug_boot|safety|class|
+------+-----+-----+-------+--------+------+-----+
| vhigh|vhigh|    2|      2|   small|   low|unacc|
| vhigh|vhigh|    2|      2|   small|   med|unacc|
| vhigh|vhigh|    2|      2|   small|  high|unacc|
| vhigh|vhigh|    2|      2|     med|   low|unacc|
| vhigh|vhigh|    2|      2|     med|   med|unacc|
| vhigh|vhigh|    2|      2|     med|  high|unacc|
| vhigh|vhigh|    2|      2|     big|   low|unacc|
| vhigh|vhigh|    2|      2|     big|   med|unacc|
| vhigh|vhigh|    2|      2|     big|  high|unacc|
| vhigh|vhigh|    2|      4|   small|   low|unacc|
+------+-----+-----+-------+--------+------+-----+
```

4. **Dodatkowe czynności sprawdzające bazę danych:**
    a. **Pogrupowanie samochodów według klas bezpieczeństwa:**

```
df.groupBy('class').count().show()
```

```
+-----+-----+
|class|count|
+-----+-----+
|unacc| 1210|
|  acc|  384|
|vgood|   65|
| good|   69|
+-----+-----+
```

    b. **Policzenie ile jest wszystkich przykładowych samochodów:**

```
df.count()
```

```
1728
```

    c. **Sprawdzenie ilości kolumn w tabeli:**

```
len(df.columns)
```

```
7
```

    d. **Sprawdzenie rodzaju danych w tabeli:**

```
df.printSchema()
```

```
root
 |-- buying: string (nullable = true)
 |-- maint: string (nullable = true)
 |-- doors: string (nullable = true)
 |-- persons: string (nullable = true)
 |-- lug_boot: string (nullable = true)
 |-- safety: string (nullable = true)
 |-- class: string (nullable = true)
```

    e. **Pokazanie szczegółów dla tej tabeli:**

```
df.describe().show()
```

```
+-------+------+-----+------------------+------------------+--------+------+-----+
|summary|buying|maint|             doors|           persons|lug_boot|safety|class|
+-------+------+-----+------------------+------------------+--------+------+-----+
|  count|  1728| 1728|              1728|              1728|    1728|  1728| 1728|
|   mean|  null| null|               3.0|               3.0|    null|  null| null|
| stddev|  null| null|0.816811769737353|1.0004343105525066|    null|  null| null|
|    min|  high| high|                 2|                 2|     big|  high|  acc|
|    max| vhigh|vhigh|             5more|              more|   small|   med|vgood|
+-------+------+-----+------------------+------------------+--------+------+-----+
```

5. **Przetwarzanie danych wejściowych do postaci odpowiedniej dla modelu uczenia maszynowego:**

   Definiujemy listę kolumn, które zawierają dane tekstowe.

```
# Definicja kolumn z danymi tekstowymi
string_cols = ["buying", "maint", "doors", "persons", "lug_boot", "safety"]
```

   Tworzymy model StringIndexer do przypisania indeksów etykietom w kolumnie "class". Wprowadzamy te indeksy do ramki danych.

```
# Tworzenie modelu StringIndexer dla etykiet
labelIndexer = StringIndexer(inputCol="class", outputCol="class_id").fit(df)
df = labelIndexer.transform(df)
```

   Tworzymy obiekty StringIndexer dla każdej kolumny z danymi tekstowymi. Te obiekty przypisują unikalne indeksy do wartości tekstowych w odpowiednich kolumnach.

```
# Tworzenie obiektów StringIndexer dla każdej kolumny
indexers = [StringIndexer(inputCol=col, outputCol=col + "_index") for col in string_cols]
```

   Tworzymy obiekt VectorAssembler, który łączy wszystkie indeksy kolumn tekstowych w jedną funkcję "features". W ten sposób przygotowujemy dane w formacie wymaganym przez algorytmy uczenia maszynowego.

```
# Tworzenie obiektu VectorAssembler
assembler = VectorAssembler(inputCols=[col + "_index" for col in string_cols], outputCol="features")
```

   Tworzymy pipeline, który zawiera etapy StringIndexer i VectorAssembler. Dopasowujemy model potoku do ramki danych, co powoduje przeprowadzenie wszystkich transformacji na danych w jednym potoku przetwarzania.

```
# Tworzenie pipeline'a
pipeline = Pipeline(stages=indexers + [assembler])
```

   Wyświetlamy przekształcone dane, aby zobaczyć rezultaty.

```
# Dopasowywanie i przekształcanie zbioru danych za pomocą pipeline'a
model = pipeline.fit(df)
transformedData = model.transform(df)
```

   W rezultacie otrzymujemy ramkę danych, w której kolumny tekstowe zostały zamienione na indeksy, a wszystkie te indeksy zostały połączone w jedną kolumnę "features".

```
# Wyświetlanie przekształconego zbioru danych
transformedData.show(10)
```

```
+------+-----+-----+-------+--------+------+-----+--------+------------+-----------+-----------+-------------+--------------+------------+--------------------+
|buying|maint|doors|persons|lug_boot|safety|class|class_id|buying_index|maint_index|doors_index|persons_index|lug_boot_index|safety_index|            features|
+------+-----+-----+-------+--------+------+-----+--------+------------+-----------+-----------+-------------+--------------+------------+--------------------+
| vhigh|vhigh|    2|      2|   small|   low|unacc|     0.0|         3.0|        3.0|        0.0|          0.0|           2.0|         1.0|[3.0,3.0,0.0,0.0,...|
| vhigh|vhigh|    2|      2|   small|   med|unacc|     0.0|         3.0|        3.0|        0.0|          0.0|           2.0|         2.0|[3.0,3.0,0.0,0.0,...|
| vhigh|vhigh|    2|      2|   small|  high|unacc|     0.0|         3.0|        3.0|        0.0|          0.0|           2.0|         0.0|[3.0,3.0,0.0,0.0,...|
| vhigh|vhigh|    2|      2|     med|   low|unacc|     0.0|         3.0|        3.0|        0.0|          0.0|           1.0|         1.0|[3.0,3.0,0.0,0.0,...|
| vhigh|vhigh|    2|      2|     med|   med|unacc|     0.0|         3.0|        3.0|        0.0|          0.0|           1.0|         2.0|[3.0,3.0,0.0,0.0,...|
| vhigh|vhigh|    2|      2|     med|  high|unacc|     0.0|         3.0|        3.0|        0.0|          0.0|           1.0|         0.0|[3.0,3.0,0.0,0.0,...|
| vhigh|vhigh|    2|      2|     big|   low|unacc|     0.0|         3.0|        3.0|        0.0|          0.0|           0.0|         1.0|[3.0,3.0,0.0,0.0,...|
| vhigh|vhigh|    2|      2|     big|   med|unacc|     0.0|         3.0|        3.0|        0.0|          0.0|           0.0|         2.0|[3.0,3.0,0.0,0.0,...|
| vhigh|vhigh|    2|      2|     big|  high|unacc|     0.0|         3.0|        3.0|        0.0|          0.0|           0.0|         0.0| (6,[0,1],[3.0,3.0])|
| vhigh|vhigh|    2|      4|   small|   low|unacc|     0.0|         3.0|        3.0|        0.0|          1.0|           2.0|         1.0|[3.0,3.0,0.0,1.0,...|
+------+-----+-----+-------+--------+------+-----+--------+------------+-----------+-----------+-------------+--------------+------------+--------------------+
```

6. **Stworzenie nowej ramki “model_df”:**

   Tworzymy nową ramkę danych o nazwie *“model_df”*, która zawiera wybrane kolumny "features", "class" i "class_id" z przekształconych danych. Wyświetlamy zawartość ramki, aby zobaczyć przekształcone cechy (indeksy tekstowe) oraz oryginalne etykiety "class" i odpowiadające im indeksy "class_id".

```
model_df = transformedData.select("features", "class", "class_id")
model_df.show(10)
```

```
+--------------------+-----+--------+
|            features|class|class_id|
+--------------------+-----+--------+
|[3.0,3.0,0.0,0.0,...|unacc|     0.0|
|[3.0,3.0,0.0,0.0,...|unacc|     0.0|
|[3.0,3.0,0.0,0.0,...|unacc|     0.0|
|[3.0,3.0,0.0,0.0,...|unacc|     0.0|
|[3.0,3.0,0.0,0.0,...|unacc|     0.0|
```

7. **Rozdzielenie danych przeznaczonych na trening i na testy:**

   Dzielimy ramkę danych model_df na dwa zbiory: training_df (zbiór treningowy) i test_df (zbiór testowy). Używamy metody randomSplit z podziałem [0.7, 0.3], co oznacza, że 70% danych zostanie przypisane do zbioru treningowego, a 30% danych do zbioru testowego. Wyświetlamy liczbę wierszy w zbiorze treningowym i testowym przy użyciu metody count(), aby zobaczyć, ile przykładów mamy w każdym zbiorze.

```
training_df, test_df = model_df.randomSplit([0.7,0.3])
print("Zbiór danych przeznaczony na trening:", training_df.count())
print("Zbiór danych przeznaczony na testy: ", test_df.count())
```

```
Zbiór danych przeznaczony na trening: 1198
Zbiór danych przeznaczony na testy:  530
```

8. **Przeprowadzenie predykcji za pomocą wytrenowanego modelu:**

   Tworzymy model klasyfikatora drzewa decyzyjnego (DecisionTreeClassifier) z etykietą ustawioną na "class_id". Model ten został wytrenowany na wcześniej przygotowanym zbiorze treningowym “training_df”. Używamy go do przewidywania etykiet dla zbioru testowego “test_df” przy użyciu metody transform. Wyświetlamy wyniki przewidywań, które zawierają kolumny z cechami “features”, rzeczywistymi etykietami “class” oraz przewidywanymi etykietami “prediction”.

```
df_classifier = DecisionTreeClassifier(labelCol="class_id").fit(training_df)
df_predictions = df_classifier.transform(test_df)
df_predictions.show(10)
```

```
+-------------------+-----+--------+--------------------+--------------------+----------+
|           features|class|class_id|       rawPrediction|         probability|prediction|
+-------------------+-----+--------+--------------------+--------------------+----------+
|      (6,[0],[1.0])|unacc|     0.0| [257.0,0.0,0.0,0.0]|   [1.0,0.0,0.0,0.0]|       0.0|
|(6,[0,1],[2.0,2.0])|unacc|     0.0| [257.0,0.0,0.0,0.0]|   [1.0,0.0,0.0,0.0]|       0.0|
|(6,[0,1],[3.0,3.0])|unacc|     0.0| [257.0,0.0,0.0,0.0]|   [1.0,0.0,0.0,0.0]|       0.0|
|(6,[0,2],[1.0,3.0])|unacc|     0.0| [257.0,0.0,0.0,0.0]|   [1.0,0.0,0.0,0.0]|       0.0|
|(6,[0,2],[2.0,2.0])|unacc|     0.0| [257.0,0.0,0.0,0.0]|   [1.0,0.0,0.0,0.0]|       0.0|
```

9. **Ocena jakości klasyfikacji:**
    a. **Accuracy - dokładność: ~86,6%**

Tworzymy obiekt MulticlassClassificationEvaluator, który będzie oceniał jakość klasyfikacji. Ustawiamy etykietę (labelCol) na "class_id" i metrykę (metricName) na "accuracy" (dokładność). Następnie oceniamy jakość klasyfikacji na podstawie przewidywań (df_predictions) przy użyciu metody evaluate.

```
df_accuracy = MulticlassClassificationEvaluator(labelCol="class_id",
                                                metricName="accuracy").evaluate(df_predictions)
df_accuracy
```

```
0.8660377358490566
```

b. Weighted Precision - precyzja ważona: ~85,6%

Tworzymy obiekt MulticlassClassificationEvaluator, który będzie oceniał jakość klasyfikacji. Ustawiamy etykietę (labelCol) na "class_id" i metrykę (metricName) na "weightedPrecision" (precyzja ważona). Następnie oceniamy jakość klasyfikacji na podstawie przewidywań (df_predictions) przy użyciu metody evaluate.

```
df_precision = MulticlassClassificationEvaluator(labelCol="class_id",
                                                 metricName="weightedPrecision").evaluate(df_predictions)
df_precision
```

```
0.8558997643126389
```

c. Area Under the ROC Curve - (AUC): ~85,8%

AUC to metryka wykorzystywana w problemach klasyfikacji, która mierzy jakość klasyfikatora na podstawie charakterystyki krzywej ROC (Receiver Operating Characteristic). Krzywa ROC przedstawia zależność między wskaźnikiem prawdziwie pozytywnych (True Positive Rate - TPR) a wskaźnikiem fałszywie pozytywnych (False Positive Rate - FPR) przy różnych progu decyzyjnym. AUC jest miarą powierzchni pod tą krzywą i daje ogólny wskaźnik jakości klasyfikatora - im wyższa wartość AUC, tym lepszy klasyfikator.

```
df_auc = MulticlassClassificationEvaluator(labelCol="class_id").evaluate(df_predictions)
df_auc
```

```
0.8576111689497312
```

d. Wyświetlenie ważności cech:

Ważność cech (features) jest to wartość przypisana do każdej cechy w modelu, która wskazuje, jak bardzo dana cecha przyczyniła się do predykcji klasy. Im większa wartość ważności cechy, tym ma większy wpływ na klasyfikację. Ten wynik jest szczególnie przydatny, gdy używamy modeli opartych na drzewach decyzyjnych, które automatycznie obliczają ważność cech na podstawie sposobu, w jaki drzewo podzieliło dane i użyło cech do podejmowania decyzji.

```
df_classifier.featureImportances
```

```
SparseVector(6, {0: 0.104, 1: 0.2016, 3: 0.3783, 5: 0.3162})
```

INTERPRETACJA WYNIKÓW

Na podstawie wyników, które otrzymaliśmy, możemy ocenić jakość naszego modelu klasyfikacji:

Accuracy (dokładność) - 86,6%: Wynik oznacza, że nasz model poprawnie sklasyfikował 86,6% przypadków testowych. Jest to miara ogólnej skuteczności modelu, ale nie uwzględnia rozkładu klas w zbiorze danych.

Weighted Precision (Precyzja ważona) - 85,6%: Wynik uwzględnia nie tylko ogólną dokładność modelu, ale także uwzględnia nierówny rozkład klas w zbiorze danych. Precyzja ważona jest obliczana na podstawie precyzji dla każdej klasy, a następnie ważona jest na podstawie liczności klas. Wynik 85,6% oznacza, że nasz model osiągnął dobrą precyzję dla poszczególnych klas, biorąc pod uwagę nierówny rozkład klas w zbiorze danych.

AUC (Area Under the ROC Curve) - 85,8%: Wynik AUC odnosi się do jakości predykcji naszego modelu na podstawie krzywej ROC. Wartość AUC pomiędzy 50%, a 100% oznacza, że model ma pewną zdolność do rozróżniania klas. W naszym przypadku, wynik 85,8% wskazuje na to, że model ma umiarkowaną zdolność do rozróżniania klas, ale może nie być idealnie skuteczny w klasyfikacji.

Ważności cech: Na podstawie wyników dotyczących ważności cech, możemy zinterpretować, jakie cechy w naszym modelu są najbardziej istotne dla dokonywania predykcji. Mogą być przydatne przy interpretacji wyników oraz optymalizacji go w przyszłości.

- **Persons (37,8%):** Ta cecha ma największą ważność w naszym modelu. Oznacza to, że liczba osób w samochodzie ma duży wpływ na predykcję klasy i jest istotnym czynnikiem decydującym o klasyfikacji bezpieczeństwa samochodu.
- **Safety (31,6%):** Ma również znaczący wpływ na predykcję klasy. Wysoka ważność szacowanego bezpieczeństwa samochodu wskazuje na duże znaczenie dla klasyfikacji.
- **Maint (20,2%):** Ta cecha ma mniejszą ważność, ale nadal wpływa na predykcję klasy. Maint odnosi się do ceny utrzymania samochodu, co może wskazywać, że stan samochodu jest czynnikiem, który należy wziąć pod uwagę przy klasyfikacji.
- **Buying (10,4%):** Ta cecha ma bardzo małą ważność w naszym modelu. Oznacza to, że cena kupna samochodu ma bardzo mały wpływ na predykcję klasy, ale nadal wciąż ma.
- **Doors i Lug_boot (~0%):** Obie te cechy mają ważność bliską 0%. Oznacza to, że liczba drzwi i rozmiar bagażnika nie mają żadnego wpływu na klasyfikację dokonywaną przez nasz model. Może to sugerować, że te cechy nie są istotne dla rozróżnienia klas w tym konkretnym problemie klasyfikacji.

Podsumowując, stworzony model ma wysoką ogólną dokładność, dobrą precyzję ważoną i umiarkowaną zdolność do rozróżniania klas na podstawie AUC. Warto jednak pamiętać, że ostateczna ocena jakości modelu zależy od kontekstu i wymagań konkretnej aplikacji lub problemu, który próbujemy rozwiązać. Z modelu wynika, że liczba osób w samochodzie jest najważniejszym czynnikiem wpływającym na predykcję klasy, a następnie kolejno szacowane bezpieczeństwo, koszt utrzymania i kupna. Natomiast liczba drzwi oraz rozmiar bagażnika nie mają żadnego wpływu na bezpieczeństwo samochodu.

WYNIKI DLA RÓŻNYCH STOSUNKÓW ZBIORU TRENINGOWEGO I TESTOWEGO

Stosunek zbioru treningowego do testowego	Accuracy	Weighted Precision	AUC	Ważność cech
10 / 90	81,9%	83,9%	82,0%	0: 18% 1: 18% 3: 38% 4: 3% 5: 24%
30 / 70	86,7%	83,3%	84,8%	0: 19% 1: 9% 3: 26% 4: 7% 5: 38%
50 / 50	87,4%	84,8%	85,9%	0: 9% 1: 20% 3: 36% 4: 3% 5: 30%
70 / 30	87,6%	85,3%	86,4%	0: 9% 1: 19% 3: 25% 4: 4% 5: 42%
90 / 10	90,4%	89,2%	89,6%	0: 9% 1: 20% 3: 37% 4: 3% 5: 29%

Im wyższe wartości *Accuracy*, *Weighted Precision* i *AUC*, tym lepsza jest wydajność modelu w klasyfikacji. Najlepsze wyniki osiągnięto dla stosunku 90/10, jednak trzeba zachować ostrożność, ponieważ model może dobrze radzić sobie tylko na małym zbiorze testowym i nie uwzględnia trudnych przypadków. Dlatego nie powinniśmy polegać wyłącznie na wynikach dla tego konkretnego stosunku, ponieważ niekoniecznie będą one reprezentatywne dla innych zbiorów danych. Najbardziej optymalny stosunek to 70/30, gdzie istnieje większa pewność, że model dobrze radzi sobie na różnych przypadkach.

Ważność cech wskazuje, które cechy mają większy wpływ na predykcje modelu. Wyższy procent oznacza większy wpływ. Wyniki pokazują, że cecha 3 i 5 mają największe znaczenie dla predykcji, podczas gdy cecha 4 ma bardzo niewielki wpływ i czasem jest nawet pomijana w wynikach. Cecha 2 nie jest uwzględniana w żadnym przypadku i nie ma wpływu na predykcje.

**Legenda:*
0 - ***buying*** - koszt zakupu
1 - ***maint*** - koszt utrzymania
3 - ***persons*** - dla ilu osób maksymalnie
4 - ***lug_boot*** - pojemność bagażnika
5 - ***safety*** - szacowane bezpieczeństwo