Adatbányászat beadandó projekt

Naive Bayes m-estimate

Bagossy Attila (V8YRAZ)

2018. május 6.

1. Bevezetés

A beadandó projekt részeként egy új RapidMiner operátort készítettem, mely egy *m-estimate* valószínűségi becslést alkalmazó Naiv Bayes-osztályozót biztosít a felhasználók számára. Az operátor forráskódja elérhető a következő címen:

https://github.com/battila7/rapidminer-naive-bayes-m-estimate

2. A forráskód

2.1. Konfigurációs állományok

Az operátor (és az azt tartalmazó plugin) megfelelő működéséhez számos konfigurációs állományra van szükség, melyek az operátor által elvárt bemenetekkel, paraméterekkel és az operátor grafikus felületen való megjelenésével kapcsolatos metaadatokat tartalmaznak. A konfigurációs állományok az `src/main/resources` mappában helyezkednek el. A továbbiakban csupán a lényegesebb állományokat ismertetem.

2.1.1. `i18n/OperatorsDocNaiveBayesm-estimate.xml`

Ebben az állományban adhatjuk meg az operátor nevét és azt a csoportot, amiben a felületen meg fog jelenni. Előbbi jelen esetben a `Naive Bayes (m-estimate)`, míg utóbbi a `Modeling`.

Habár ebben a fájlban megadható az operátor és a csoport neve több nyelven is, a projekt részeként csupán az angol nevek kerültek hozzáadásra.

2.1.2. `groupsNaiveBayesm-estimate.properties`

A különböző modelleket létrehozó operátorok a RapidMinerben általában egy zöldes színnel rendelkeznek. Annak érdekében, hogy az új operátor konzisztens megjelenéssel bírjon, ebben a fájlban ugyanezt a színt rögzítettem.

2.1.3. `OperatorsNaiveBayesm-estimate.xml`

A modelleket képző operátorok közös jellemzője a színen felül a villanykörtére hasonlító ikon. Az említett fájlban ezen ikon hozzárendelése történik az operátorhoz.

2.2. Java osztályok és interfészek

A következőkben a plugin és az operátor tényleges implementációját adó Java típusokat tekintjük át. A típusok sorrendje követi az azok között fennálló függőségeket.

2.2.1. `battila.rapidminer.extension.PluginInitNaiveBayes`

Minden egyes RapidMiner pluginnak tartalmaznia kell egy inicializáló osztályt, melynek metódusait a RapidMiner elinduláskor hívja meg *reflection* segítségével.

A Naiv Bayes-osztályozóhoz nincsen szükség semmilyen egyedi inicializáló kódra, emiatt a megfelelő függvények törzsei teljesen üresek.

2.2.2. `battila.rapidminer.extension.operator.mestimate.NaiveBayesMEstimate`

A `NaiveBayesMEstimate` osztály írja le az operátort. Ugyanezen osztály kiterjeszti az `AbstractLearner` osztályt, melyet a RapidMiner fejlesztői kifejezetten a gépi tanulást implementáló operátorok ősosztályának szántak. Az említett osztály kiterjesztésével ugyanolyan viselkedést érhetünk el, mint a beépített operátorok, azaz az operátorunk rendelkezni fog egy, az adathalmaz számára fenntartott bemeneti porttal, valamint olyan kimeneti portokkal, mint például a létrehozott modell vagy az eredeti adathalmaz. Ezen felül számos olyan kódrészletet és metódus-implementációt tartalmaz, mely az ilyen típusú operátorok esetén azonos, egyszerűsítve ezzel a fejlesztők dolgát.

A legfontosabb metódus, melyet a klienseknek (így a `NaiveBayesMEstimate` osztálynak is) implementálnia kell, a `public Model learn(ExampleSet exampleSet)`, mely egy, az adathalmazra illeszkedő modellt állít elő. Esetünkben ez a metódus egy `NaiveBayesModel` példányt fog visszaadni.

```
1 @Override
2 public Model learn(ExampleSet exampleSet) throws OperatorException {
3     return new NaiveBayesModel(
4         exampleSet,
5         getParameterAsDouble(M_PARAMETER),
6         retrievePriorValues(exampleSet));
7 }
```

Kódrészlet 1. A `learn` metódus implementációja

Az operátor paraméterezését a `public List<ParameterType> getParameterTypes()` függvény megvalósításával befolyásolhatjuk. Az m-estimate valószínűségi becslés alkalmazásához két paraméterre van szükség, az m értékekére valamint az egyes osztályokhoz tartozó *a priori* valószínűségi értékekre.

```
1  types.add(new ParameterTypeDouble(
2      M_PARAMETER,
3      "This parameter defines the m value used when calculating the probability.",
4      Double.MIN_VALUE,
5      Double.MAX_VALUE,
6      DEFAULT_M_VALUE
7  ));
8
9  types.add(new ParameterTypeList(
10     PRIOR_PARAMETER,
11     "Defines the a priori (estimated) probabilities of the individual classes.",
12     new ParameterTypeString(
13             PRIOR_CLASS_PARAMETER,
14             "A possible class."
15     ),
16     new ParameterTypeDouble(
17             PRIOR_PROBABILITY_PARAMETER,
18             "The probability of the class.",
19             0,
20             1
21     ),
22     false
23 ));
```

Kódrészlet 2. A paraméterek definiálása

2.2.3. `battila.rapidminer.extension.operator.mestimate.NaiveBayesModel`

A Naiv Bayes-osztályozó két legfontosabb tevékenységét, azaz a tanulást és az osztályozást is a `NaiveBayesModel` valósítja meg, mely a RapidMiner beépített `PredictionModel` osztályát terjeszti ki.

A tanulási folyamat a valószínűségi becslések kiszámítását jelenti, mely közvetlenül az osztály egy példányának létrehozásakor megtörténik. A becslések meghatározása attribútumonként történik, az attribútumokhoz rendelt `ProbabilityCalculator` objektumok segítségével. Pontosabban, a folytonos értékeket felvevő attribútumokhoz egy `GaussianProbabilityCalculator`, míg a no-

minális értékeket felvevő attribútumokhoz egy `NominalProbabilityCalculator` objektum kerül létrehozásra. Természetesen a különleges szereppel (*role*) rendelkező attribútumok nem lesznek figyelembe véve.

```
final Map<String, ProbabilityCalculator.Builder> calculatorBuilders = Arrays.stream(regularAttributes)
    .collect(toMap(Attribute::getName, attribute -> {
        if (attribute.isNominal()) {
            return new NominalProbabilityCalculator.Builder(attribute, m, priors);
        }
        return new GaussianProbabilityCalculator.Builder(attribute);
    }));
```

Kódrészlet 3. A valószínűségi becsléseket meghatározó `ProbabilityCalculator` objektumok létrehozása.

Miután a fent említett objektumok létrejöttek, következik a bemeneti adathalmaz feldolgozása. A feldolgozás során egyfelől rögzítésre kerül, hogy az egyes osztályokba hány rekord tartozik, másfelől az egyes attribútumok értékeivel frissítésre kerülnek a megfelelő `ProbabilityCalculator` objektumok is.

```
for (Example example : trainingExampleSet) {
    countPerClass.merge(example.getValue(labelAttribute), 1, Integer::sum);

    for (Attribute attribute : regularAttributes) {
        calculatorBuilders.get(attribute.getName()).add(example);
    }
}
```

Kódrészlet 4. A bemeneti adathalmaz feldolgozása.

Az osztályozás a `performPrediction` metódusban történik. A metódus egy adathalmazt kap, melyet el kell látni címkékkel. A metódus ehhez egyszerűen végigfut a bemenet minden egyes rekordján, és meghatározza az adott rekordhoz tartozó címkét, valamint egy konfidencia értéket.

```
@Override
public ExampleSet performPrediction(ExampleSet exampleSet, Attribute predictedLabel) {
    final Attribute[] regularAttributes = exampleSet.getAttributes().createRegularAttributeArray();

    for (Example example : exampleSet) {
        final Prediction prediction = predictExample(example, regularAttributes);

        example.setValue(predictedLabel, prediction.predictedClass);

        prediction.confidenceMap.forEach(example::setConfidence);
    }

    return exampleSet;
}
```

Kódrészlet 5. Adathalmaz rekordjainak osztályozása.

Egy rekord osztályozása a következő képlet alapján történik:

$$\hat{y} = \underset{k \in \{1,\dots,K\}}{\arg\max}\ P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k)$$

ahol C_k a k-adik osztály, $P(C_k)$ az ehhez az osztályhoz tartozó *a priori* valószínűség, a produktum pedig az egyes attribútumokhoz tartozó feltételes valószínűségeken fut végig. Az osztályozó azt a C_k osztályt fogja választani a rekord címkéjeként, melyhez a legnagyobb valószínűség tartozik. Az ezt megvalósító forráskód a következő:

```
private Prediction predictExample(Example example, Attribute[] regularAttributes) {
    double maxProbability = Double.MIN_VALUE;
    double summedProbability = 0;
    double predictedClass = 0;

    final Map<String, Double> confidenceMap = new HashMap<>();

    for (Map.Entry<Double, Integer> clazzEntry : countPerClass.entrySet()) {
        double clazzProbability = priors.get(clazzEntry.getValue().doubleValue());

        for (Attribute attribute : regularAttributes) {
            clazzProbability *= Optional.ofNullable(calculators.get(attribute.getName()))
                    .map(calculator ->
                        calculator.calculateFor(example.getValue(attribute), clazzEntry.getKey()))
                    .orElse(Double.MIN_VALUE);
        }

        summedProbability += clazzProbability;

        confidenceMap.put(labelAttribute.getMapping().mapIndex(clazzEntry.getKey().intValue()),
                          clazzProbability);

        if (clazzProbability > maxProbability) {
            maxProbability = clazzProbability;
            predictedClass = clazzEntry.getKey();
        }
    }

    final double heyCompilerThisOneIsFinal = summedProbability;
    confidenceMap.replaceAll((clazz, probability) ->
        isNaN(probability) ? 0.0 : (probability / heyCompilerThisOneIsFinal));

    return new Prediction(predictedClass, confidenceMap);
}
```

Kódrészlet 6. Rekord osztályozása.

Jól látható, hogy a kód egy maximumkereséső ciklus köré szerveződik, amely végigfut a lehetséges osztályokon. A legnagyobb valószínűséget a `maxProbability`, a kiválasztott osztályt pedig a `predictedClass` változók tartalmazzák. Egy adott osztályhoz tartozó valószínűség kiszámítása a következő módon zajlik. Először hozzárendeljük a `clazzProbability` változóhoz az osztály *a priori* valószínűségét. Ezután sorra szorzunk az attribútumokhoz tartozó feltételes valószínűségekkel. Amennyiben az így kapott érték nagyobb, mint az eddigi maximális valószínűség, akkor módosítjuk a maximális értéket és a kiválasztott osztályt. A címke meghatározásán túl, e metódusban konfidencia értékeket is kiszámítunk, melyek a `confidenceMap` változóban kerülnek tárolásra; minden osztályhoz egy valós érték. A metódus visszatérési értéke egy, a kiválasztott osztályt és a konfidencia értékeket tartalmazó `Map`-et magában foglaló `Prediction` objektum lesz.

2.2.4. `battila.rapidminer.extension.operator.mestimate.NominalProbabilityCalculator`

A `NominalProbabilityCalculator` osztály számítja ki nominális attribútumokra a valószínűségi becsléseket. Egy adott attribútumértékre a következő módon határozza meg a becslést:

$$P(x_i \mid C) = \frac{N_{ic} + mp}{N_c + m},$$

ahol N_{ic} a C osztályba tartozó, az adott attribútumértékkel rendelkező rekordok száma, N_c az összes C osztálybeli rekord száma, m egy felhasználó által megadott érték, p pedig a C osztály *a priori* valószínűsége.

Implementációt tekintve, a `NominalProbabilityCalculator` osztály `calculateFor` metódusa felelős a valószínűség kiszámításáért. A metódus kap egy attribútumértéket, valamint egy osztályt, és visszaadja az ezekhez számolt valószínűséget. Ehhez a tanulási folyamat során meghatározott értékeket használja fel, azaz azt, hogy egy adott attribútumértékhez hány adott osztálybeli rekord tartozott, valamint, hogy attribútumoktól függetlenül, hány adott osztálybeli előfordulás volt.

```java
@Override
public double calculateFor(double attributeValue, double clazz) {
    final int count = valueCountPerClass.get(attributeValue).get(clazz);

    final double numerator = (double)count + m * priors.get(clazz);

    return numerator / (countPerClass.get(clazz).doubleValue() + m);
}
```

Kódrészlet 7. Nominális attribútum feltételes valószínűségének kiszámítása.

2.2.5. battila.rapidminer.extension.operator.mestimate.GaussianProbabilityCalculator

Folytonos attribútumokhoz tartozó valószínűségi becsléseket a `GaussianProbabilityCalculator` osztály képes számolni. Ahogy az osztály neve is mutatja, feltesszük, hogy az attribútum normális eloszlású, majd pedig a bementi adathalmaz alapján megbecsüljük az eloszlás paramétereit.

Amikor számítanunk kell egy adott feltételes valószínűséget, akkor vesszük az adott osztályhoz tartozó paramétereket, és azok alapján becsüljük az attribútumértékhez és az osztályhoz tartozó feltételes valószínűséget. A paramétereket a `distributionPropertyMap` tartalmazza, melyet a tanulási fázis során töltünk fel értékekkel.

```java
@Override
public double calculateFor(double attributeValue, double clazz) {
    final DistributionProperties props = distributionPropertyMap.get(clazz);

    final double pow = -(Math.pow(attributeValue - props.mean, 2) / (2 * props.variance));

    final double rat = 1.0 / Math.sqrt(2 * Math.PI * props.variance);

    return rat * Math.exp(pow);
}
```

Kódrészlet 8. Nominális attribútum feltételes valószínűségének kiszámítása.

3. Az operátor használata

3.1. Az operátor telepítése

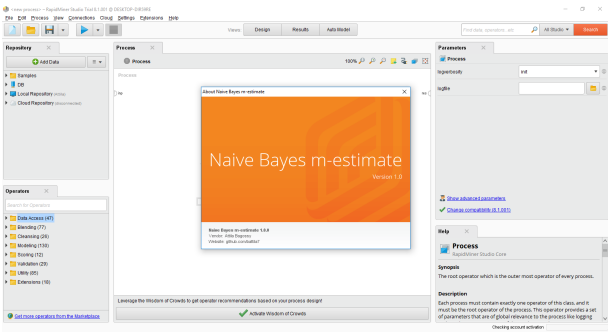
Az operátor telepítése előtt győződjünk meg arról, hogy a RapidMiner valamely verziója, valamint a Java Development Kit (legalább 8-as verziójú) telepítve van a számítógépünkre. Az operátor kódját tartalmazó repository klónozásához gitre is szükség van.

A telepítéshez, a repository klónozását követően, adjuk ki a következő parancsot annak gyökérkönyvtárában:

```
./gradlew installExtension
```

Kódrészlet 9. Az operátor telepítése.

Ha a telepítés sikeres volt, akkor a RapidMiner elindítását követően, az *Extension* menüpont *About Installed Extensions* almenüjében meg fog jelenni egy *About Naive Bayes m-estimate Extension* menüpont.


Naive Bayes m-estimate
Version 1.0