Data Mining und Maschinelles Lernen

TECHNISCHE UNIVERSITÄT DARMSTADT

Projekt Teil 3 - Abgabe bis 30.01.2018 – 23:59

Ziel des Projektes ist es, praktische Erfahrungen im Maschinellem Lernen zu sammeln. Hierzu sollen mehrere Projektaufgaben mit Hilfe des Machine Learning Frameworks *Weka* (`http://www.cs.waikato.ac.nz/ml/weka/`) gelöst werden. Des Weiteren folgt im späterem Verlauf eine Implementierungsaufgabe. Das Projekt soll in Kleingruppen von Studierenden bearbeitet werden. Die Anmeldung, Bildung von Gruppen und Abgabe finden online in Moodle statt (`https://moodle.informatik.tu-darmstadt.de/course/view.php?id=346`). Die Abgaben erfolgen in Moodle bis zu den jeweiligen Stichtagen. Folgendes ist zu den Abgaben zu beachten:

- **Format:** Die Abgabe soll eine einzelne **Präsentation** im **PDF-Format** sein (z.B. 4:3 Querformat). Der Umfang einer Abgabe sollte 15 Folien nicht überschreiten.
- **Inhalt** Die Präsentation soll **selbsterklärend** sein. Das bedeutet, dass die Lösung ohne mündliche Erklärung nachvollziehbar ist. Das Dokument muss genügend Erläuterungen und Ausführungen enthalten. Eine reine Ansammlung von Grafiken und Tabellen ohne jegliche Begleittexte ist hierfür nicht ausreichend. Die **Bewertung** findet allein anhand der PDF-Datei statt (abgesehen von der Implementationsaufgabe).
- **Abbildungen** Tabellen, Diagramme, etc. müssen **vollständig beschriftet** sein, d.h. sie müssen zumindest direkt an der Abbildung eine kurze Beschreibung enthalten und ausreichend kommentiert sein. Insbesondere sollte bei Abbildungen von Resultaten der verwendete Datensatz angegeben werden.

Für die Programmieraufgabe ist der Java-Quellcode als *einzelne* NearestNeighbor.java einzureichen (keine GIT/SVN Links). Die Beantwortung der Implementationsfragen erfolgt wie oben beschrieben als PDF-Dokument.

Für einen Klausurbonus ist es nicht zwingend nötig alle Aufgaben zu bearbeiten. Bei Teilabgaben kann noch ein entsprechender Teilbonus erreicht werden. Das Implementationsprojekt umfasst etwa 1/3 der zu erreichenden Punkte. Der Aufwand sollte ca 10 Stunden/Person betragen.

In den jeweiligen Aufgaben verwenden Sie eine vorgegebene Anzahl von Datensätze, die Sie der Sammlung von Klassifikations- und Regressionsdatensätzen unter `http://www.ke.informatik.tu-darmstadt.de/lehre/ws-16-17/mldm/projekt/datasets.zip` entnehmen, falls die Datensätze nicht in der Aufgabenstellung festgelegt werden. Achten Sie hierbei bitte darauf, möglichst unterschiedliche Datensätze zu wählen und diese in den einzelnen Aufgaben zu variieren. Bei der Auswahl der Datensätze ist weiterhin zu beachten, dass bestimmte Lernverfahren nicht mit allen Datensätze umgehen können. Optional können Sie dieses Problem beheben, indem Sie die Daten vorverarbeiten, z.B. mittels `FilteredClassifier` und einem entsprechendem Preprocessing-Filter. In diesem Fall, bzw. falls Sie die Standardeinstellungen in Weka modifizieren, geben Sie dies bitte in ihrer Lösung an.

Aufgabe 1 Ensemble-Lernen (3 Punkte)

In dieser Aufgabe sollen unterschiedliche Ensemble-Methoden eingesetzt und deren Ergebnisse verglichen werden. Der Entscheidungsbaumlerner J48 soll als Basislerner verwendet werden. Nutzen Sie für diese Aufgabe die Datensätze `labor`, `yeast` und `car`.

a) Bestimmen Sie die Genauigkeit des regulären J48.

b) Verwenden Sie nun Bagging mit J48 und AdaBoost mit J48. Benutzen Sie außerdem noch Random Forests. Bestimmen Sie für die so erhaltenen Klassifizierer die Genauigkeiten für eine stetig wachsende Anzahl von Iterationen (bei den Random Forest verändern Sie bitte die Anzahl der Bäume). Wie interpretieren Sie die Entwicklung der erzielten Genauigkeiten?

Aufgabe 2 Pre-Processing (2 Punkte)

Wählen Sie drei Klassifikationsdatensätze aus. Erstellen Sie für jeden Datenset eine diskretisierte Version unter Verwendung des Filters `weka.filters.unsupervised.attribute.Discretize`.

a) Schätzen Sie die Genauigkeit von `J48` mittels Cross-validation auf den ursprünglichen Daten und auf den diskretisierten Daten ab.

b) Wie interpretieren Sie den Vergleich der Genauigkeiten und der Größe der gelernten Bäume dieser drei Experimente (die Ergebnisse können über die drei Datensets gemittelt werden)?

Aufgabe 3 Implementierung von kNN (12 Punkte)

Im Rahmen dieser Aufgabe werden Sie einen k-Nearest Neighbor Klassifizierer implementieren, wie er auch in der Vorlesung und Übung vorgestellt wird. Hierzu wird ein Framework bereitgestellt, welches Kompatibilität mit WEKA gewährleistet. Bei korrekter Implementation sind die Resultate des Klassifizieres identisch mit der WEKA-Version von IBk. Neben dem Framework werden auch JUnit TestCases zur Verfügung gestellt mit welchen die korrekte Funktionsweise überprüft werden kann.
Die Aufgaben Aufgabe 3.1 und Aufgabe 3.2 sind fast ausschließlich mit Informationen aus dem Vorlesungsskript zu lösen, wohingegen Aufgabe Aufgabe 3.3 etwas Transferleistung und eigene Überlegungen erfordert.

Installation

Das Projekt ist in Java geschrieben und es wird empfohlen, Eclipse[1] als IDE zu verwenden. Das Projektarchiv enthält entsprechende Eclipse Projektsignaturen. Zudem wird JUnit 4 benötigt, welches aber bereits Bestandteil der Eclipse Java Development Tools[2] ist. Die Anbindung an WEKA [3] kann entweder über das Hinzufügen der Datei weka.jar zum classpath erfolgen oder indem man den Quellcode als Projekt einbindet. Das Projekt wurde für WEKA 3.7.11 geschrieben, ist aber auch mit WEKA 3.7.10 folgend kompatibel (getestet bis 3.7.12).

Abgabe

Die Abgabe erfolgt zusammen mit der letzten Abgabe des Projektes. Die Fragen dieser Aufgabe werden im Abgabe-PDF mit den anderen Fragen beantwortet, nicht direkt im Code. Die Abgabe des Quellcodes besteht nur aus der **NearestNeighbor.java**. Weitere Dateien oder Links finden keine Berücksichtigung. Die Verwendung von weiteren Klassen oder Libraries ist also nicht möglich. Im Code sollen zudem keine Annahmen über die Reihenfolge des Ausführens der Methoden gemacht werden. Z.b. wird zu Testzwecken determineManhattanDistance ausgewertet, ohne dass vorher learnModel aufgerufen wird.

Das Framework

Zu implementieren ist die Klasse tud.ke.ml.project.classifier.NearestNeighbor. Diese erbt von tud.ke.ml.project.framework.classifier.ANearestNeighbor, welche ein lightweight Framework definiert so wie das Handling der Konfiguration. Die Klasse weka.classifiers.lazy.keNN definiert die Schnittstelle zu WEKA und darf genauso wie die abstrakte Basisklasse nicht modifiziert werden.

Instanzen sind im Framework als Liste von Objekten gespeichert. Ein jedes Objekt ist entweder ein String, und repräsentiert in diesem Falle ein nominales Attribut, oder ein Double, welches für ein numerisches Attribut steht. Das

[1] `https://www.eclipse.org/`
[2] `http://www.eclipse.org/jdt/`
[3] `http://www.cs.waikato.ac.nz/ml/weka/`

Klassenattribut ist immer nominal. Der Index des Klassenattributes entspricht dem Rückgabewert von getClassAttribute und muss nicht immer dem letzten Attribut entsprechen.

Gestartet wird das Projekt über tud.ke.ml.project.main.RunWEKA. Der neue Classifier ist nun in allen Teilen des Programmes unter lazy.keNN zu finden. Die Klasse main.SimpleRun ist ein einfaches Beispiel, welches während der Entwicklung genutzt werden kann, falls Sie nicht immer die WEKA GUI verwenden wollen. Im Package tud.ke.ml.project.junit befinden sich zudem die erwähnten JUnit-Tests. junit.SimpleValidation und junit.AdvancedValidation testen verschiedene Konfigurationen des Klassifizieres. Sollten diese Tests alle erfolgreich durchlaufen, ist dieser sehr wahrscheinlich korrekt implementiert.

Zu allen relevanten Methoden ist eine Javadoc-Dokumentation vorhanden.

JUnit JUnit-Tests können ausgeführt werden, indem die entsprechende Library dem Projekt hinzugefügt wird. In Eclipse ist diese bereits vorhanden und muss daher nicht mehr zusätzlich hinzugefügt werden. Zum Zesten muss eine JUnit-Java-Datei ausgewählt werden und per Kontextmenü *Run As→JUnit Test* ausgeführt werden. Rote Kreuze bedeuten, dass der Test eine Exception ausgelöst hat. Blaue Kreuze repräsentieren einen fehlgeschlagenen JUnit-Test. Für eine genauere Beschreibung von JUnit siehe `http://www.vogella.com/tutorials/JUnit/article.html`.

Die Aufgabe Im Folgenden ist zu beachten, dass zum Testen alle Methoden implementiert sein müssen. Es ist aber natürlich möglich, hierfür erst einmal nur Dummy-Methoden zu verwenden, die einfach einen fixen Wert zurückgeben.
Definieren sie zuerst getMatrikelNumbers, da diese Funktion für die finale Bewertung verwendet wird!
Achtung: Geben Sie hier die **Matrikelnummern** der Gruppenteilnehmer als konkatenierten String im Format "XXXXXXXX,YYYYYYYY,ZZZZZZZZ"(also z.B. "12322323,134534334,15244324") zurück.

Aufgabe 3.1 Der Basis-Klassifizierer

a) Implementieren Sie die Methode learnModel, welche die als Argument übergebenen Trainingsdaten intern speichert. Des Weiteren soll die Methode vote implementiert werden, indem Sie getUnweightedVotes aufrufen. Die daraus resultierende Map muss an getWinner übergeben werden um die aus den Votes resultierende Klasse zu bestimmen.

b) getNearest muss für alle Instanzen des gespeicherten Modells die Manhattan-Distanz zur übergebenen Test-Instanz berechnen. Dies soll mit Hilfe der zu implementierenden Methode determineManhattanDistance erfolgen. Die Liste, die zurückgegeben wird, muss auf die nächsten getkNearest Elemente beschränkt sein. Die Liste beeinhaltet Pair Objekte, bestehend aus der Instanz sowie dem Abstand.

c) Wenn Sie nun noch die Methode getWinner implementieren, sollten Sie eine erste, lauffähige Implementation haben. Die genannte Methode soll den Name/Wert der Klasse mit den meisten Votes zurückgeben.

Aufgabe 3.2 Inverse Distance Weighting und Euclidean Distance

a) Implementieren Sie nun getWeightedVotes, welches die Stimmen als Summe der inversen Distanzen berechnet. vote muss diese Methode in Abhängigkeit von isInverseWeighting aufrufen.

b) Genauso muss die Methode getNearest in Abhängigkeit von getMetric die euklidsche Distanz als Entfernungsmaß nutzen. Hierzu soll die Methode determineEuclideanDistance genutzt und implementiert werden.

Aufgabe 3.3 Normalization and Tie-Breaking

a) Es ist meistens sinvoll, die Attributwerte zu normalisieren. Dafür soll die Methode normalizationScaling die nötigen Skalierungs- und Translationsfaktoren zurückliefern, abhängig von isNormalizing. Die Werte sollen dabei in das Intervall [0, 1] normalisiert werden. In der Methode getNearest muss nun das resultierende Array aufgeteilt und als protected double[] scaling für die Skalierung, bzw. protected double[] translation für die Verschiebung gespeichert werden, wobei für jedes Attribut (auch das potentielle Klassenattribut) ein Eintrag vorhanden sein muss. Für nominelle Attribute wird der Eintrag in den Arrays ignoriert, d.h. sie können leer gelassen werden. Diese Arrays **müssen** nun genutzt werden, um die Attributwerte entsprechend zu normalisieren, d.h. das Resultat von normalizationScaling darf nicht direkt übernommen werden!

b) In der Methode getWinner kann es vorkommen, dass mehr als eine Klasse die meisten Stimmen bekommen hat. Implementieren Sie hierfür eine Entscheidungsfunktion und begründen Sie diese. (1 Punkt)

c) In der Methode getNearest kann es passieren, dass mehrere Instanzen den gleichen Abstand haben. Wie sollte man hierbei vorgehen? Begründen und implementieren Sie ihre Methode. (1 Punkt)