Machine Learning 101

Alexander Bresk

May 12, 2017

Revision: 3 (2017-05-12)

Einführung

Das Dokument soll als Merkzettel dienen, welches die relevantesten Vorgehensweisen im Bereich Machine Learning zusammenfasst. Alle Sektionen bauen aufeinander auf und sollten bei einer Machine Learning Anwendung auch in dieser Reihenfolge durchgeführt werden: Entscheidung für den Typ der Anwendung (Sektion 1), das besorgen der Daten (Sektion 2), das Erstellen der Feature-Representation (Sektion 3), Prototyping des Feature-Vektors (Sektion 4), die Auswahl eines Verfahrens für die Implementation (Sektion 5) und das anschließende Messen der Ergebnisse (Sektion 6).

Machine Learning wird in vielen Unternehmen eingesetzt, um anhand von statistischen oder logischen Algorithmen zu versuchen, Entscheidungen schneller und basierend auf vorhandenen Daten zu treffen. Dieses Dokument befasst sich ausschließlich mit **statistischen Methoden**. Das hat den Grund, dass in Firmen, in denen große Datenmengen anfallen, meist statistische Modelle, welche Unschärfe und Fehlertoleranz lernen können, eingesetzt werden können.

Hat man bereits bestimmte Vorstellungen, wozu man die Daten nutzen möchte und es existieren Zielwerte oder Klassen, setzt man Supervised Learning ein. Im Gegensatz dazu existieren Algorithmen, die sich unter Unsupervised Learning zusammenfassen lassen. Diese Algorithmen versuchen die Daten selbstständig zu separieren (Clustering).

Learning 1 Bei Big-Data-Anwendungen nutzt man statistische Methoden, um Daten zu separieren. Logische Regel Sets eignen sich nicht sehr gut, da sie nur mit viel Aufwand in der Lage sind, Unschärfe abzubilden.

Learning 2 Es gibt eine Pipeline, nach der man in den meisten Fällen vorgehen sollte und die dabei hilft, den Prozess des Machine Learning zu verbildlichen.

Learning 3 Supervised Learning wird benutzt, wenn man Datensätze und einen zu lernenden Zielwert definieren kann. Ist das nicht der Fall, kann man Unsupervised Learning nutzen, um Daten zu clustern und sich ein Überblick über die Struktur der Daten zu verschaffen.

1 Klassifikation vs. Regression

Neben einigen weiteren Methoden zur Anwendung von Machine Learning sind Klassifikation und Regression die beiden Populärsten. Das sind die wesentlichen Unterschiede:

Klassifikation / Klassifikator Bei der Klassifikation versucht man eine Menge von Daten in verschiedene Klassen zu separieren. Die Daten, die man versucht zu trennen, gehören mehreren Klassen an (die sich gegenseitig ausschließen). Der Klassifikator versucht, diese Daten zu trennen.

Regression / Regressor Die Regression versucht die Daten einem Schätzwert zuzuordnen. Im Gegensatz zur Klassifikation möchte man die Daten nicht in verschiedene Bereiche teilen, sondern einen bestimmten Wert vorhersagen.

Beispiel Wir wollen Vorhersagen, wie sich der Aktienkurs einer beliebigen Aktie verhält. Wir haben Daten gesammelt und trainieren einen Regressor. Den Regressor würden wir dazu bringen, den Börsenkurs direkt zu schätzen. Einen Klassifikator würden wir dazu bringen, zwischen den Klassen $\{up, down\}$ zu entscheiden (Tendenz des Kurses).

Learning 4 Es gibt zwei wichtige Formen, wie man das Ergebnis von einem Machine Learning Experiment ausdrücken kann. Die Klassifikation zeigt die Konfidenz aller Klassen. Im Gegensatz dazu schätzt die Regression einen bestimmten Wert.

2 Was sind eigentlich diese Daten?

Nach dem man entschieden hat, ob man einen Klassifikator oder einen Regressor implementieren will, kommt es darauf an, die richtigen Daten zu bereitzustellen sowie die richtigen Labels zu ermitteln. Ein Label beschreibt den Datensatz. Der Vorgang der Label-Bestimmung nennt sich Annotation. Ein Beispiel:

```
24,android,male,payer
18,ios,male,nopayer
```

In diesem Fall sind die Daten durch ein Komma separiert. Die Datensätze zeigen das Alter, Plattform, Geschlecht und das Label, ob der User bereits bezahlt hat oder nicht. Mit diesem Datenset würde sich ein Klassifikator trainieren lassen, der zwischen den Klassen $\{payer, nopayer\}$ entscheidet.

```
24,android,male,17.99
18,ios,male,0.0
43,ios,male,30.00
```

Mit diesem Datenset könnte man einen Regressor trainieren, der anhand der Features schätzen soll, wie hoch der CLV[1] eines Users ist. Der Regressor sowie Klassifikator kann mit solchen Datensets trainiert werden. Nutzt man ihn nach dem Training, ist natürlich kein Label vorhanden. Das Verfahren hat gelernt und schätzt das Label für ein gegebenes Datenset.

[1] Customer Lifetime Value

Learning 5 Daten benötigen eine Annotation (ein Label oder Zielwert), damit eine Maschine weiß, was sie lernen muss. Diese Zielwerte können Klassenbezeichnungen ($\{payer, nopayer\}$) oder reellwertige Angaben sein $(21.99, 17.99)$

3 Vom Feature zum Vektor

Nachdem man die passenden Datensätze ausgewählt hat und diese mit einem Label versehen hat, folgt der nächste Schritt. Die Daten müssen nun in einen Vektor transformiert werden: $\{_{0,1,0,0,0,1,0,1,0,0,0,0,1,1}\}$. So könnte ein möglicher Feature-Vektor aussehen. Die Anzahl der Features (durch das Komma separierte Zahlenwerte), die man für die Darstellung der Daten im Vektor benutzt, nennt man Dimensionen.

Dieser Schritt sollte vom Engineer verstanden werden. Der Manager muss lediglich wissen, dass es diesen Schritt gibt und welche Probleme damit einher gehen können:

- Wählt man eine ungünstige Codierung der Features, kann es passieren, dass sich die Features nicht gut separieren lassen.
- Ist die Codierung während Trainings-, Test-, Validations- und Production-Phase nicht konsistent, erhält man keine konsistenten Ergebnisse.

Das Programm das aus den Daten die Feature-Vektoren formt, nennt sich Feature-Extraktor.

$$\{24, anda, male\} \rightarrow \{0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1\} \quad (1)$$

$$\{18, ios, male\} \rightarrow \{1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1\} \quad (2)$$

Wie wir in den zwei Darstellungen 1 und 2 sehen können, existieren für die beiden verschiedenen Datensätze zwei verschiedene Feature-Vektor-Darstellungen. Datensätze mit identischen Attributen hätten auch die gleiche Darstellung im Feature-Raum.

Learning 6 Für das Machine Learning benötigt man einen Feature-Vektor, der das Feature so codiert, dass es ein Algorithmus separieren kann. Deshalb muss der Feature-Vektor für verschiedene Klassen oder Zielwerte auch verschiedene Werte

aufweisen. Ist das nicht der Fall, kann der der Algorithmus auch nicht lernen.

Learning 7 Bei dem Schritt der Feature-Extraktion können diverse Fehler auftreten. Feature-Darstellungen sollten also immer konsistent sein.

4 Prototyping

Nachdem man die Vektordarstellungen vorbereitet hat, kann man entweder gleich einen Klassifikator oder Regressor trainieren und validieren oder aber man testet seine Feature-Repräsentation in einer Prototyping-Umgebung. Das ist besonders sinnvoll, wenn man besonders schnell testen will, ob sich die Daten mit der gefundenen Repräsentation separieren lassen. Für diesen Schritt empfehlen sich die Programme:

- KNIME[2]
- RapidMiner
- Weka[3]

Mit diesen Programmen kann man ohne Programmierkenntnisse versuchen, seine Vektordarstellungen klassifizieren zu lassen. Das hilft, um herauszufinden, wie gut eine Vektordarstellung ist.

Learning 8 Mittels bestimmten Programmen kann man die Feature-Vektoren testen. Dabei benötigt man keinerlei Programmierkenntnisse sondern nur ein grobes Verständnis der Algorithmen.

5 Bekannteste Verfahren

Um die Klassifikation bzw. Regression durchzuführen, gibt es verschiedene Methoden. Hier werden die populärsten mit ihren Vor- und Nachteilen genannt.

Naive Bayes Naive Bayes ist ein sehr simples Verfahren, welches auf dem Bayes-Theorem basiert. Der größte Vorteil daran ist, dass man Naive Bayes ohne aufwändiges Modell-Training online lernen lassen kann. Allerdings ist Naive Bayes ein sehr simples Verfharen, weshalb es auch nicht für die besten Resultate bekannt ist.

Neuronales Netz Ein neuronales Netz trainiert Gewichte, um die Eingangsdaten auf die Zielwerte zu mappen (Fehlerminimierung). Ein neuronales Netz eignet sich für Big-Data-Anwendungen, da die Anzahl der Features und die Anzahl der Gewichte (die zu optimierenden Parameter) konstant bleibt - auch bei großen Datenmengen. Es kann beim Training allerdings sehr lange dauern.

Decision Tree Ein Entscheidungsbaum scannt die Daten und versucht, anhand des Attributes, das am meisten Entropie minimiert und am meisten Informationszugewinn bringt, die Daten zu trennen. Dies wiederholt der Baum in jedem Schritt, bis er alle Attribute verarbeitet hat.

Random Forest Ein Random Forest besteht aus mehreren Entscheidungsbäumen, die jeweils mit einem disjunkten Set von Trainingsdaten (jeder Baum kriegt ein anderes) trainiert werden. Dadurch erhält man ein ausgewogeneres Ergebnis.

Support Vector Machine Support Vector Maschinen versuchen die Vektoren in einem n-dimensionalen Vektorraum mithilfe von Kernelfunktionen so zu platzieren, dass die Schätzung möglichst korrekt wird. Die SVM[4] ist für Big-Data Anwendungen nicht geeignet. Es existieren allerdings abgewandelte Formen der SVM, die die Anzahl der gespeicherten Support-Vektoren zu minimieren.

Learning 9 Es existieren verschiedene Algorithmen, die diverse Vor- und Nachteile haben und sich für bestimmte Anwendungen besser eignen als andere. Prinzipiell sind sie aber vergleichbar.

6 Messen und Einschätzen

Nachdem Training wird gemessen, wie gut das Verfahren die Daten verarbeitet hat. Dafür existieren

[2] Konstanz Information Miner
[3] Framework der Universität von Waikato, Neuseeland.
[4] Support Vector Machine

verschiedene Metriken, um den Erfolg des Verfahrens zu messen. Für die Klassifikation werden Metriken für die binäre Klassifikation benutzt. Existieren mehrere Klassen, wird die binäre Klassifikation in Klasse a gegen alle anderen Klassen gemessen. Existieren beispielsweise 3 Klassen, so wird, um die Metrik für Klasse a zu erzeugen, Klasse a gleich *Positive* und die Klassen b, c gleich *Negative* gesetzt.

6.1 Klassifikation

True Positive Der Klassifikator hat einen Datenpunkt der Klasse Positive richtig erkannt.

True Negative Der Klassifikator hat einen Datenpunkt der Klasse Negative richtig erkannt.

False Positive Der Klassifikator hat einen Datenpunkt der Klasse Negative als Datenpunkt der Klasse Positive erkannt.

False Negative Der Klassifikator hat einen Datenpunkt der Klasse Positive als Datenpunkt der Klassse Negative erkannt.

Accuracy Die Genauigkeit. Wie viele Datenpunkte wurden richtig klassifiziert.

F-Measure Ähnlich wie die Accuracy. Arbeitet man auf einem unausgewogenen Datenset, bei der eine Klasse über viel mehr Datenpunkte als eine andere verfügt (Spammer vs. Non-Spammer oder Payer vs. Non-Payer), bringt die F-Measure mit einem harmonischen Mittel mehr Information in die "Genauigkeitsmessung".

Learning 10 Es existieren verschiedene Metriken, um den Erfolg eines Klassifikators zu messen. Sie basieren auf binärer Klassifikation. Benutzt man ein, durch die Anzahl der Repräsentanten der Klassen, unausgeglichenes Datenset, so sollte man, um die Genauigkeit zu messen, die F-Measure benutzen.

6.2 Regression

Da man bei der Regression keine Schätzung einer Klasse sondern einen Zahlenwert erhält, fällt die Beurteilung des Regressors anders aus. Man hat verschiedene Möglichkeiten:

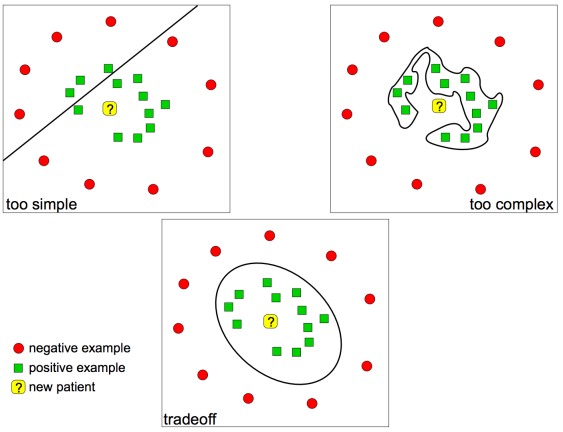
Binäres Thresholding Man wählt eine Grenze, die aussagt, wie groß die Abweichung vom Wert sein darf. Ist der geschätzte Wert innerhalb der Grenze, entscheidet man sich für *Positive*. Befindet sich der Wert außerhalb der Grenze, entscheidet man sich für *Negative*. Dadurch lassen sich wiederum die Metriken der binären Klassifikation anwenden. Man kann also die Metriken aus Sektion 6.1 nutzen.

Fehlerklassen Eine Alternative dazu sind Fehlerklassen. Diese Fehlerklassen können den Grad der Ungenauigkeit definieren. So erreicht man bei der Beurteilung des Regressors eine bessere Granularität. Je weiter die Schätzung vom Zielwert entfernt ist, desto weniger Qualität hat die Fehlerklasse.

Learning 11 Es gibt mehrere Möglichkeiten einen Regressor zu beurteilen. In den meisten Fällen kann man die Metriken der binären Klassifikation wählen. Fehlerklassen sind ebenfalls ein beliebtes Mittel. Welche Option man wählen sollte, hängt von der Art der Anwendung ab.

7 Over- und Under-Fitting

Beim Training von Klassifikatoren existiert ein sehr populäres Problem. Man möchten einen Klassifikator lernen, der während der Trainingsphase so gut wie möglich performt. Dieser Klassifikator sollte aber nach der Trainingsphase in der Lage sein, in der Testphase sowie in Produktion, gute Ergebnisse zu erzählen. Er soll also die Trainingsdaten lernen, ohne sie auswendig zu lernen.



Spricht man vom auswendig lernen, so meint man meist das Overfitting. Ein Klassifikator der 100% der Daten richtig klassifiziert, hat sie auswendig gelernt. Im Gegensatz dazu kann ein Klassifikator auch underfitten. Das bedeutet, der Klassifikator nutzt einen zu einfachen Ansatz, um die Daten zu separieren.

Learning 12 Over- und Under-Fitting des Klassifikators ist zu vermeiden, da der Klassifikator das Training-Set entweder auswendig lernt oder einen zu einfachen Ansatz wählt, um die Daten zu separieren.

8 Take aways

Hier werden die Take aways aus diesem Dokument zusammengefasst:

- Bei Big-Data-Anwendungen nutzt man statistische Methoden, um Daten zu separieren. Logische Regel Sets eignen sich nicht sehr gut, da sie nur mit viel Aufwand in der Lage sind, Unschärfe abzubilden.
- Es gibt eine Pipeline, nach der man in den meisten Fällen vorgehen sollte und die dabei hilft, den Prozess des Machine Learning zu verbildlichen.
- Supervised Learning wird benutzt, wenn man Datensätze und einen zu lernenden Zielwert definieren kann. Ist das nicht der Fall, kann man Unsupervised Learning nutzen, um Daten zu clustern und sich ein Überblick über die Struktur der Daten zu verschaffen.
- Es gibt zwei wichtige Formen, wie man das Ergebnis von einem Machine Learning Experiment ausdrücken kann. Die Klassifikation zeigt die Konfidenz aller Klassen. Im Gegensatz dazu schätzt die Regression einen bestimmten Wert.
- Daten benötigen eine Annotation (ein Label oder Zielwert), damit eine Maschine weiß, was sie lernen muss. Diese Zielwerte können Klassenbezeichnungen ($\{payer, nopayer\}$) oder reellwertige Angaben sein $(21.99, 17.99)$
- Für das Machine Learning benötigt man einen Feature-Vektor, der das Feature so codiert, dass es ein Algorithmus separieren kann. Deshalb muss der Feature-Vektor für verschiedene Klassen oder Zielwerte auch verschiedene Werte aufweisen. Ist das nicht der Fall, kann der der Algorithmus auch nicht lernen.
- Bei dem Schritt der Feature-Extraktion können diverse Fehler auftreten. Feature-Darstellungen sollten also immer konsistent sein.
- Mittels bestimmten Programmen kann man die Feature-Vektoren testen. Dabei benötigt man keinerlei Programmierkenntnisse sondern nur ein grobes Verständnis der Algorithmen.
- Es existieren verschiedene Algorithmen, die diverse Vor- und Nachteile haben und sich für bestimmte Anwendungen besser eignen als andere. Prinzipiell sind sie aber vergleichbar.
- Es existieren verschiedene Metriken, um den Erfolg eines Klassifikators zu messen. Sie basieren auf binärer Klassifikation. Benutzt man ein, durch die Anzahl der Repräsentanten der Klassen, unausgeglichenes Datenset, so sollte man, um die Genauigkeit zu messen, die F-Measure benutzen.
- Es gibt mehrere Möglichkeiten einen Regressor zu beurteilen. In den meisten Fällen kann man die Metriken der binären Klassifikation wählen. Fehlerklassen sind ebenfalls ein beliebtes Mittel. Welche Option man wählen sollte, hängt von der Art der Anwendung ab.
- Over- und Under-Fitting des Klassifikators ist zu vermeiden, da der Klassifikator das Training-Set entweder auswendig lernt oder einen zu einfachen Ansatz wählt, um die Daten zu separieren.

Anhang: Prototyping

Wie bereits in Sektion 4 angesprochen, kann man zur Erstellung eines Prototyps das Programm KNIME[5] nutzen. Nachfolgend ist eine beispielhafte Pipeline abgebildet.

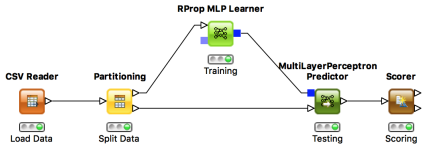


Figure 1: Beispiel einer Pipeline in KNIME um ein neuronales Netz zu trainieren.

Zu erst wird die Datei mit den Feature-Vektoren mit dem CSV Reader gelesen. Diese Datei hat diese oder eine ähnliche Form:

24,1,0,1,0,1,1,12,8.1,1,1,payer
31,0,0,1,0,1,1,12,4.9,1,1,payer
18,0,1,1,1,0,1,7,8.7,0,0,nonpayer

Hier ist jeder Feature-Vektor mit dazugehörigen Klassen-Label in einer Zeile repräsentiert. Nach dem Einlesen werden diese Datenpunkte dann partitioniert. Beispielsweise werden hier 80% der Daten zum Training und die restlichen 20% der Daten zum Testen benutzt. Nach dem Training wird das trainierte Modell zusammen mit den Testdaten in den Predictor geschoben. Der Predictor schätzt ein Label für die Testdaten. Schließlich wird der Scorer dafür genutzt, um eine Statistik über die Güte des Klassifikators zu erstellen. Diese Statistik ist nachfolgend beschrieben: Dabei

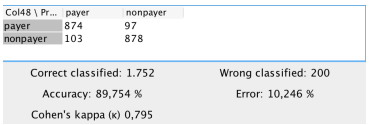


Figure 2: Resultate eines Klassifikators in KNIME.

baut KNIME eine sogenannte Confusion-Matrix auf. Diese Matrix sagt aus, wie viele Datenpunkte die ursprünglich zur einer Klasse gehörten, von Klassifikator auch richtig eingeschätzt wurden. Neben der Klassifikation mit den neuronalen Netz

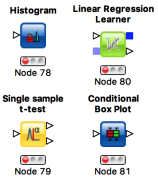


Figure 3: Verschiedene Möglichkeiten in KNIME.

existieren noch weitere nützliche Operationen, die KNIME wertvoll machen.

[5]Konstanz Information Miner