learning machines

data science for sustainability

Accelerating Machine Learning on FMCG Data

Für effiziente Entwicklung von KI-Anwendungen im Einzelhandel

12.08.2020

Jochen Sautter, Dr. Boris Lau

Learning Machines GmbH
Gerberau 9a
79098 Freiburg

Inhalt

Machine Learning im Einzelhandel

Getrieben durch den schnell wachsenden Onlinehandel, Personalisierung, technologische Innovationen und regulatorische Anforderungen durchläuft die Retail-Branche derzeit einen rasanten Wandel und einen Digitalisierungsschub. Hierbei ergeben sich vielfältige Potentiale für den Einsatz von Machine Learning (ML) - Technologien. Diese erfolgreich zu nutzen wird künftig ein zunehmend bedeutender Erfolgsfaktor.

Allerdings erfordert die Umsetzung von ML Projekten relativ viele Ressourcen in Form von qualifizierten Data Science Experten. Dies gilt bereits für Machbarkeitsstudien, um zu testen, ob und mit welcher Performance gegebene Daten überhaupt verwertbare Prognosen erlauben. Ein gesamter Projektzyklus von der Idee bis zum produktiven Betrieb erfordert typischerweise mehrere Personenmonate.

Der wirtschaftlich erfolgreiche breitere Einsatz von Machine Learning hängt daher wesentlich davon ab, dass es gelingt die Entwicklungszyklen für solche Modelle deutlich zu verkürzen. Die gute Nachricht ist, dass es hierfür starke Möglichkeiten gibt.

Use Cases für ML auf Basis von Artikeldaten

Das Gesamtsortiment großer Retail-Unternehmen umfasst heute mehrere Hunderttausend bis über eine Million Artikel. Die Verwaltung dieser Artikelstammdaten, mit oftmals mehreren hundert Attributen verschiedener Formate, stellt somit eine große Herausforderung dar. Der Einsatz von ML-Methoden eröffnet zum Einen vielfältige Möglichkeiten zu Kostensenkungen durch KI-Unterstützung beim Management dieser Daten, zum Anderen, um in diesen Daten verborgene werthaltige Informationen für vielfältige Anwendungen zu erschließen.

Der wirtschaftlich erfolgreiche breitere Einsatz von Machine Learning Methoden hängt wesentlich davon ab, dass es gelingt, die Entwicklungszyklen für solche Modelle deutlich zu verkürzen. Die gute Nachricht ist, dass es hierfür starke Möglichkeiten gibt.

Ein Großteil der Retail-Unternehmen arbeitet auf Basis von Artikeldaten des **GDSN-Datenpools**, wie sie von der atrify GmbH im XML Format zur Verfügung gestellt werden. Diese Situation ist ein repräsentatives Beispiel für die Herausforderungen, die die Entwicklung von ML-Applikationen stellt, und Ausgangspunkt der nachfolgenden exemplarischen Betrachtungen zu Use Cases und den Möglichkeiten der Automatisierung und Effizienzsteigerung bei deren Umsetzung.

Prädiktion von Artikeleigenschaften
Artikeln werden in der unternehmensinternen Datenhaltung i.d.R. viele weitere Attribute und Klassifizierungen zugeordnet, die in den GDSN-Daten nicht oder nur unvollständig enthalten sind. Dies betrifft z.B. Filterkategorien für Online-Suchen, Informationen, die durch Lebensmittel-Apps abgefragt werden können, die Logistik betreffende Eigenschaften wie Lagertemperaturen, Stapelhöhen etc., Informationen zur Haltbarkeit oder regulatorisch relevante Angaben (z.B. LMIV).

Diese Zuordnungen können entweder zeitaufwändig manuell erfolgen, oder mit eigens programmierten Skripts, die diese aus den bekannten Attributen ableiten. Dies ist jedoch oft sehr komplex oder führt zu schwachen Ergebnissen.

In vielen Fällen kann ein Machine Learning Modell mit “gelabelten” Artikeln, für welches diese gewünschten Attribute vorliegen, trainiert werden und das Attribut dann für weitere Artikel vorhersagen. Da ML-Modelle auch sehr komplexe Zusammenhänge lernen können, ist dieser Trainingsprozess nicht nur einfacher, sondern führt oft zu weit besseren Ergebnissen als die manuelle Spezial-Programmierung.

KI - unterstütztes Labeling von Daten

Gelegentlich werden neue Attribute eingeführt, die irgendwie aus den bekannten Attributen abgeleitet werden können. Dann muss ein gesamter Datenbestand “from scratch” mit diesem Attribut versehen werden. Dieser Labeling-Prozess kann von einen Klassifikationsmodell unterstützt werden, das fortlaufend mit den bereits gelabelten Daten trainiert wird. Ein solches Modell kann bereits nach rund 10 Beispielen erste Lerneffekte aufweisen, und nach 100 bis 1.000 Artikeln bereits gute Vorschläge machen, die das manuelle Labeling immer besser unterstützen können. Am Ende dieses Prozesses steht ein trainierter Klassifikator, der dann auch auf zukünftige Artikel angewendet werden kann.

Suche nach Anomalien zur Datenpflege

Methoden des sogenannten “Unsupervised Learnings” benötigen für das Training gar keine Labels (korrekte Vorhersagen), sondern nutzen alleine die in den Artikeldaten enthaltenen Strukturen. Ein solches Modell kann z.B. abschätzen, in welchem Ausmaß die Attributkombination eines Artikels “ungewöhnlich” ist, und ist somit ein sehr universelles Tool zur Fehlersuche.

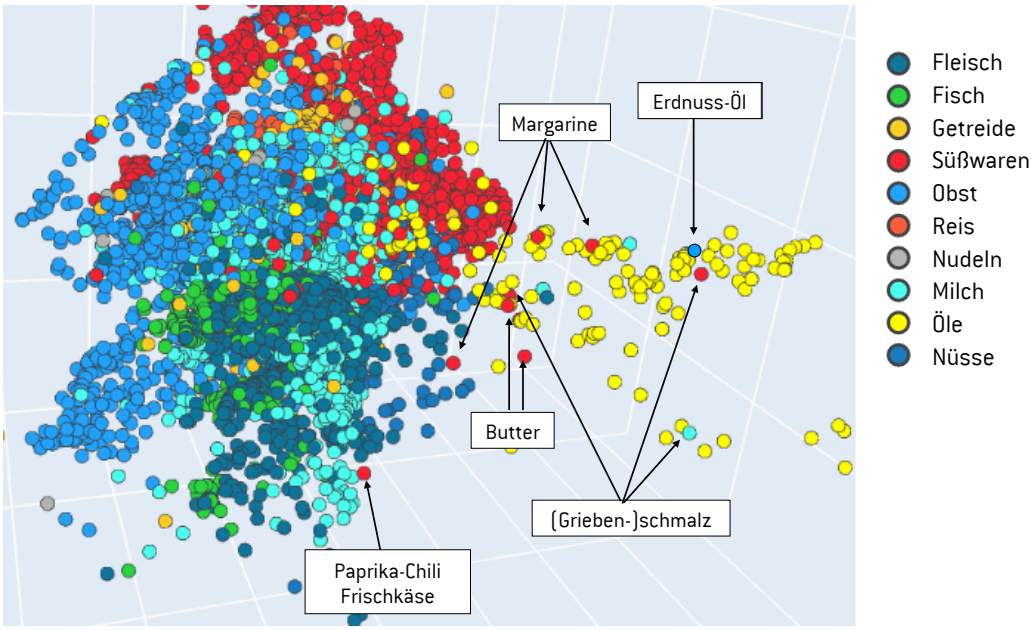

Abb. 1: Ausschnitt aus einer Spezial-Visualisierung unter Verwendung von Unsupervised Learning, die Lebensmittel-Artikel mit ähnlichen Nährwerten räumlich benachbart abbildet. Die einzelnen Punkte, sind nachträglich mit Kategorien eingefärbt, die beim Training des Modells nicht verwendet wurden. “Ausreißer” in dieser Darstellung erweisen sich oft als Fehlklassifizierungen in den Daten (im Bild einige fälschlich als “Süßwaren”, *Obst” oder “Milch” klassifizierte Artikel). Die hier gelernten Repräsentationen können außer zur Fehlersuche auch zur Bestimmung von Ähnlichkeiten verwendet werden, für das Training von Supervised Modellen, für die nur wenige Labels verfügbar sind oder wie hier für Visualisierungen.

Suche nach ähnlichen Artikeln, Clustering
Ein anderes Beispiel für unsupervised Learning ist, ein universelles Maß für die Ähnlichkeit zwischen Artikeln zu lernen und Artikeln nach Gruppen (Clustern) zu sortieren. Die Suche nach ähnlichen Artikeln könnte z.B. beim personalisierten Marketing helfen, oder um ein einem Sortiment Ersetzungen vorzunehmen.

Use Cases auf Basis weiterer Daten
Die Kombination der Artikeldaten mit weiteren Daten, wie historischen Preis- und Absatzzahlen, Kundenprofilen, kalendarischen Daten, Wetterdaten, Standort-Daten u.A. ermöglicht eine Vielzahl weiterer Use Cases:

- Optimierung der Preisgestaltung und Rabattaktionen
- Personalisierte Marketingaktionen mit Apps online und vor Ort
- Optimierung von Lagerhaltung und Logistik
- Automatisierung von Sichtprüfungen
- Standortspezifische Sortimentauswahl
- zeitlich und räumlich hoch aufgelöste Absatzprognosen
- Auswahl von Standorten für Filialen

AutoML: Automated Machine Learning

Die meisten Machine Learning Lösungen werden heute mit den etablierten Open Source Tools wie Pandas Numpy, scikit learn, Tensorflow oder Pytorch in Python realisiert. Diese Bibliotheken werden fortlaufend weiterentwickelt.

Ausgehend vom Base Case, der relativ anspruchsvollen Programmierung mit diesen Tools, ergeben sich verschiedene Möglichkeiten der Vereinfachung und Effizienzsteigerung durch Automatisierung.

AutoML-Technologien zur Hyperparameter Optimierung
Machine Learning Modelle weisen eine Anzahl von frei wählbaren “Hyperparametern” auf, von deren optimaler Einstellung ihre Performance abhängt. Der Begriff “AutoML” hat sich etabliert für die automatisierte Lösung dieser Optimierungsaufgabe. AutoML ist Gegenstand lebhafter Forschungs-aktivitäten und akademischer Challenges. Das betrifft insbesondere die Neuronalen Netze, die als die leistungsfähigsten und vielseitigsten ML-Modelle gelten, jedoch auch die meisten Hyper-Parameter und Variablen in der Modellarchitektur aufweisen, und damit eine besonders anspruchsvolle Optimierungsaufgabe stellen.

AutoML-Technologien sind als Open Source Bibliotheken verfügbar, können also in Projekte integriert werden. State-of-the-art sind hier z.B. NASnet (Neural Architecture Search) und ENAS von Google Brain, sowie das an der Uni Freiburg entwickelte BOHB (Bayesian Optimization and HyperBand), das bei vielen Aufgaben ähnlich gut performed wie NASnet, jedoch mit weit geringerer Rechenkapazität auskommt.

Diese leistungsfähigen AutoML-Tools automatisieren jedoch nur einen Bruchteil des gesamten Produktzyklus. Einer Faustformel in der Machine Learning Community zufolge verbringt ein DataScientist etwa 10% seiner Zeit mit seiner Lieblingsbeschäftigung, der Erstellung und Konfiguration von ML

Einer Faustformel in der ML Community zufolge verbringt ein Data Scientist etwa 10% seiner Zeit mit seiner Lieblingsbeschäftigung, der Erstellung und Konfiguration von ML Modellen, und 90% mit dem Zusammentragen, Aufbereiten, und Transformieren von Daten verschiedener Herkunft und Formate, sowie mit Deployment und Integration.

Modellen, und 90% mit dem Zusammentragen, Aufbereiten, und Transformieren von Daten verschiedener Herkunft und Formate, sowie mit Deployment und Integration.

Vollautomatische AutoML Systeme
Es gibt leistungsfähige Systeme, wie z.B. Dataiku, DataRobot oder Google AutoML (alle USA), die gesteuert durch GUIs vollautomatisch Modelle erstellen und trainieren, und auch modellnahe Teile des Data Preprocessing übernehmen. Solche Modelle können eine Vielzahl auch anspruchsvollerer Problemstellungen sehr schnell lösen und erfordern im Prinzip keine Data Science Spezialisten.

Die Performance der so erzeugten Modelle ist in vielen Fällen gut, auch wenn nicht davon ausgegangen werden kann, dass sie alle Optimierungspotentiale ausschöpfen, die einem Experten zur Verfügung stehen. Sie stoßen zudem an Grenzen beim Umgang mit komplexen strukturierten Daten. Die Integration der so erzeugten Modelle in Software Landschaften ist teilweise möglich, teilweise durch die Lizenzbedingungen erschwert.

Der bedeutendste Nachteil solcher Lösungen ist, dass der hohe Automatisierungsgrad erkauft wird mit einem Mangel an Flexibilität. Während vollautomatische Lösungen in “Lehrbuchumgebungen” verblüffend schnell gute Ergebnisse liefern, wird es in meist nicht lehrbuchartigen Real-World Situationen schnell aufwändig oder unmöglich, passend zu machen was nicht passend ist. Gerade größere Unternehmen werden oft davor zurückschrecken, sich beim strategischen Aufbau von ML-Ökosystemen in das Korsett einer solchen Lösung zu begeben, und stattdessen eigene maßgeschneiderte Ressourcen aufbauen.

Für Unternehmen gilt es die Balance zu finden zwischen der vollen Flexibilität der Lösungsentwicklung from scratch und dem Automatisierungs-potential höher aggregierter Umgebungen.

Open Source-basierte modulare Bibliotheken
Eine Alternative zu vollautomatisierten Lösungen sind modulare Bibliotheken kohärenter high-level Klassen auf Basis der etablierten Open Source Technologien. Sie erfordern Programmierung durch Data Science Experten, realisieren jedoch einen großen Teil der nachfolgend diskutierten Automatisierungspotentiale. Zugleich bleiben sie flexibel und offen für Spezialentwicklungen auf Seiten des Data-Preprocessing, der Modelle sowie der Integration in IT-Landschaften. Sie profitieren zudem von der rasanten Weiterentwicklung der verfügbaren Open Source Tools.

Manche besonders IT-affine Unternehmen entwickeln selbst solche Open Source-basierten Ökosysteme einschließlich domänenspezifischer Werkzeuge.

Beispiele für Basis-Toolboxen sind Kortical (UK), H20 (US), oder unsere Learning Machines Engine (Deutschland).

Entwicklungszyklus eines ML Modells für GDSN-Artikeldaten

Dieser Abschnitt sowie der nächste zu Design Choices sind insbesondere für technologieaffine Leser und Data Scientists interessant. Wier erläutern wir am Beispiel eines Klassifikators die typischen Prozessschritte eines ML-Projektes auf Basis von GDSN-Artikeldaten und den jeweils zu erwartenden Aufwand. Wir erläutern Möglichkeiten zur Automatisierung und schätzen die hierdurch erzielbaren Einsparungspotentiale ab.

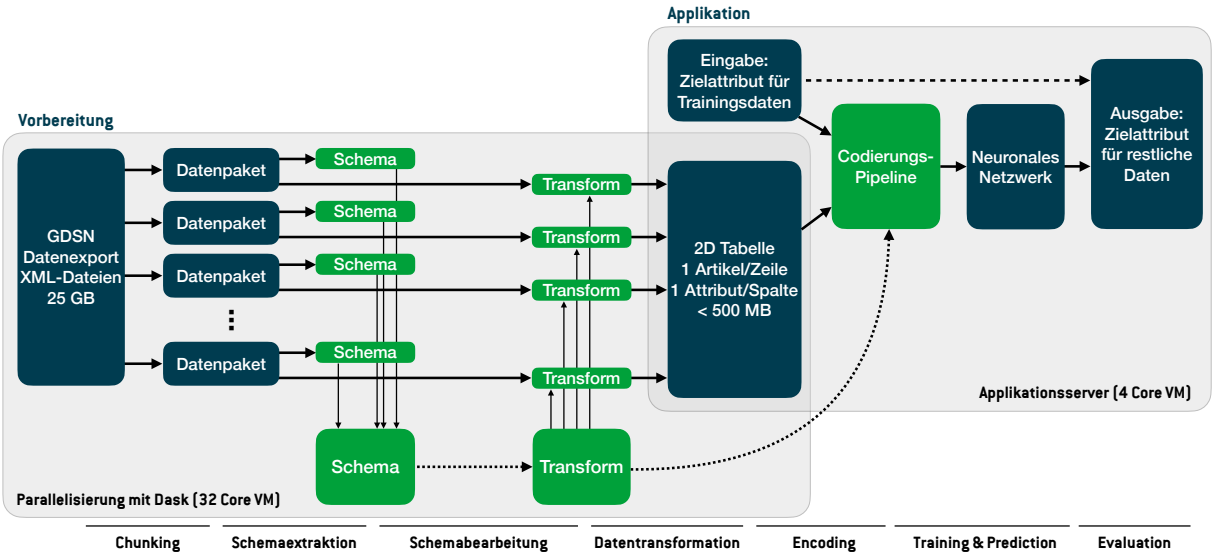

Abb. 2: Pipeline von den XML Rohdaten bis zum trainierten Klassifikator, der hier als Neuronales Netz realisiert ist.

1) ETL: Extraktion / Transformation / Laden der Daten

Die Rohdaten liegen in einer Vielzahl von XML Dateien vor, die aus stark verschachtelten Attributbäumen bestehen und auch Listenfelder variabler Länge enthalten können. Die ML-Modelle benötigen die Daten jedoch meistens in Form einer zweidimensionalen numerischen Tabelle. Dies erfordert im ersten Schritt eine komplexe Datentransformation auf Basis eines die Daten beschreibenden Schemas. Oft gibt es ein vordefiniertes Schema in Form von XSD-Dateien, welches jedoch nur die theoretisch erlaubten Datenstrukturen und Attribute beschreibt, nicht aber deren tatsächliche Nutzung im vorliegenden Fall. Um die Daten in ein nutzbares Format zu transformieren, sollte ein Schema erstellt werden, dass den tatsächlich vorgefundenen Daten entspricht (Schema on Read).

1a) Schema-Extraktion

Ein Modul zur datengetriebenen Schemaextraktion generiert zunächst ein kohärentes Schema, das auf alle Eingabedaten passt. Hierbei wird neben den Attributnamen und deren Datentyp auch der Füllgrad jedes Attributs (in wie vielen Fällen ist es vorhanden bzw. fehlt es?), die Kardinalität von Listenfeldern (wieviele Einträge kommen typischerweise vor?) und weitere Metadaten erfasst.

Dieser Prozess läuft vollautomatisch ab und erfordert keinen menschlichen Input. Bei einem großen Datenvolumen kann dieser Vorgang durch Parallelisierung, z.B. mit Hilfe von Dask oder Apache Spark elegant skaliert werden.

1b) Schemabearbeitung
Bevor das so ermittelte Schema zur Transformation der Daten in eine 2D-Tabelle verwendet werden kann, sind oft noch Anpassungen erforderlich. Hierbei sind Listenfelder mit variabler Länge eine besondere Herausforderung, da diese im Rahmen der automatischen Datentransformation in eine Tabellenstruktur mit fester Länge überführt werden müssen. Je nach Semantik der Daten bietet sich die Beschränkung auf die ersten oder häufigsten N Elemente an, oder eine "Entfaltung" anhand ausgewählter Unterattribute, d.h. für jeden möglichen Eintrag der Liste wird eine eigene Spalte, mit dem Listeneintrag als Suffix des Spaltennamens angelegt.

Ein XML Schema definiert die Struktur eines Datensatzes, etwa Name und Datentypen aller erlaubter Attribute.

1c) Datentransformation
Unter Verwendung des so gewonnenen Schemas können nun die kompletten Eingabedaten in eine zweidimensionale Tabelle überführt werden. Auch eine Filterung nach bestimmten Attributen lässt sich hierbei leicht realisieren. Dieser Prozess kann wiederum vollautomatisch ablaufen.

Die resultierende 2D Tabelle im Apache Parquet Format hat nur noch einen Bruchteildes Datenvolumens der ursprünglichen XML-Dateien. Für ca. 1 Million Artikel benötigt unsere Implementation einer derartigen Pipeline weniger als eine Stunde Rechenzeit, bei parallelisierter Verarbeitung in einer Cloud VM mit 32 oder mehr CPU Cores, und Dask oder Apache Spark als Parallelisierungs-Framework.

Automatisierungspotential
Während die Schritte a) und c) voll automatisiert werden können, erfordern die Anpassungen des Schemas in b) vom Entwickler oder kundigen Endnutzer einige Designentscheidungen, etwa zur Sprachauswahl und den Attribut-Schlüsseln, entlang derer die Entfaltung der Listen erfolgen soll.

Ein von Learning Machines speziell entwickeltes Tool unterstützt die gesamte Prozesskette a) bis c), wobei die erforderlichen Entscheidungen in Schritt b) mit einfachen high-level Kommandos formuliert werden können.

Einsparungspotential durch ETL Pipeline
Mit der zu großen Teilen automatisierten Erstellung einer derartigen ETL-Pipeline wird der andernfalls hohe Entwicklungs- und Pflegeaufwand auf einen Bruchteil reduziert.

2) Daten-Codierung
Die bisherigen Prozessschritte haben die Daten in ein zweidimensionales Tabellenformat gebracht, ohne die eigentlichen Werte zu verändern. Für die Verarbeitung in Machine Learning Modellen müssen die Daten der einzelnen Spalten jedoch je nach Datentyp in geeigneter Weise in rein numerische Formate codiert werden, wie hier z.B. für neuronale Netze erläutert.

Kategoriale Daten werden mit One Hot Encoding codiert, also als Vektor mit einer binären Variablen je vorkommender Klasse. **Textfelder** mit Namen,

Kurzbeschreibungen, Zutatenlisten etc. können mit verschiedenen Verfahren zu einem Vektor fester Länge codiert werden. Diese umfasst auch eine linguistische Vorverarbeitung, u.a. mit Entfernung von Stopwörtern, Wortstammerkennung und Zerlegung von Kompositas. **Numerische Daten** sollten, je nach Typ des gewählten ML Modells, normiert werden. **Bilddaten** werden in zweidimensionale Bitmaps transformiert und skaliert. **Audiodaten** können verschiedenen Preprocessing Schritten unterworfen werden, z.B. durch Fourier-Transformation in ein Power-Spektrum. Weitere Transformationen beinhalten die **Komprimierung** numerischer Arrays, die weit überwiegend aus Nullen bestehen, durch Repräsentation als "Sparse Matrizen", die Entfernung von **Duplikaten**, das Zusammenführen von Daten verschiedener Herkunft, und schließlich die **Aufteilung** der Daten in unabhängige Trainings-, Validierungs- und Test-Sets für eine aussagekräftige Auswertung.

Für alle diese Transformationen gibt es leistungsfähige Open Source Tools, z.B. von scikit-learn, Pandas und Numpy im Python Universum. Eine Datencodierung mit insgesamt rund 10 bis 20 solcher Schritte erfordert hierbei schnell mehrere 100 Codezeilen. Der Aufwand, mit Test, Debugging, etc. liegt bei 3-10 Tagen.

Viele dieser Codierungen müssen mit den tatsächlichen Daten initialisiert ("gefitted") werden, etwa um bei einem Text Vectorizer dessen Vokabular zu bestimmen, oder bei einem OneHot Encoder die Liste der möglichen Klassen. Die gefitteten Codierer müssen für die spätere Verwendung in der Applikation gespeichert, ihre Versionen verwaltet werden.

Codierungs-Pipelines

Es empfiehlt sich, die einzelnen Codierungsschritte in einer Pipeline zu organisieren. Eine solche Pipeline von Codierern kann u.A. folgendes leisten:

- Datenintegrität: Sicherstellen, dass die strikte Trennung von Trainings- und Validierungsdaten auch für die Pipeline gilt
- Die gesamte gefittete Codierungsstrecke kann in einer einzigen Datei gespeichert werden, hierdurch vereinfachtes und fehlersicheres Versionsmanagement
- die Codierung kann paketweise erfolgen (Mini-Batching), um auch Daten transformieren zu können, die nicht in den Speicher passen
- Caching: einmal codierte Daten können zwischengespeichert werden.

Scikit-learn bietet ein framework für eine solche Pipeline, allerdings ohne Mini-Batching. Auch Googles Tensorflow bietet Klassen zum Bau von Pipelines.

Automatisierungspotential

Höher aggregierte Toolboxen wie die Learning Machines Engine bringen die verschiedenen Open Source Klassen in ein kohärentes System von Codierern mit einheitlichen APIs, die dann von übergeordneten Routinen aufgerufen und effizient zu Pipelines kombiniert werden können, und zudem auch Schnittstellen bieten für eine automatisierte Konfiguration der darauf aufbauenden ML Modelle (s.u.).

Einsparpotentiale durch High-Level Frameworks

- schnellere und weniger fehleranfällige Programmierung mit high-level Kommandos für viele Standard Codierer
- bessere Debug-Informationen bei Fehlkonfigurationen

- schnelleres und fehlersicheres Experimentieren durch smartes Caching in der Pipeline (bei großen Datenmengen kann die Transformation z.B. von Textdaten langwierig sein)

Im Vergleich zur Programmierung from scratch lassen sich die Einsparungspotentiale durch geeignete high-level Frameworks mit 60-80% einschätzen.

3) Machine Learning Modell und Training

Wir wählen hier als Model ein Multi-Channel Neuronal Network, das besonders leistungsfähig Daten verschiedener Herkunft und Formate gemeinsam prozessieren kann (s.u. zur Modellauswahl). Erstellung und Training eines solchen Modells kann in 20 bis 50 Zeilen Code in Tensorflow 2.0 (Keras style) programmiert werden. Der Aufwand mit Tests und Debugging kann mit 1-3 Tagen abgeschätzt werden.

Automatisierung

Bei vollautomatischen Systemen erfolgt die Erstellung und Konfiguration der Modelle automatisch, sollte aber auch manuell angepasst werden können.

Bei Einsatz modularer high-level Tools kann die Erstellung und das Training eines Neuronalen Netzes auch mit komplexerer Architektur mit je einem Befehl erfolgen. Die Dimensionierung erfolgt zum Einen datengetrieben automatisch, zum Anderen können die gewünschte Topologie und Parametrisierung des Modells sowie des Trainingsprozesses in JSON Formaten definiert werden. Ein trainiertes Modell kann incl. seiner ganzen Parametrisierung gespeichert, und damit einfach reproduziert werden. Die Parametrisierung in high-level Formulierungen vereinfacht zudem die Optimierung mit AutoML-Verfahren (nächster Punkt).

Einsparpotentiale durch High-Level Frameworks

Die größte Zeitersparnis durch high-level Bibliotheken liegt in der geringeren Fehleranfälligkeit einer solchen Programmierung als in nativem Tensorflow oder PyTorch, und dürften bei mindestens 50% liegen.

4) Modelloptimierung

Die Leistungsfähigkeit und enorme Vielseitigkeit von Neuronalen Netzen wird erkauft mit einer sehr viel aufwändigeren Parametrisierung. Neben der Performance determiniert die Parametrisierung auch die Größe und damit Schnelligkeit des Modells. Der Aufwand für die Suche nach optimalen Parametern wächst im Prinzip exponentiell mit der Zahl dieser Parameter. Da jeder Test einer Parameterkonfiguration einen Trainings- und Validierungslauf des Netzwerks erfordert, wächst dieser Prozess schnell ins Uferlose.

Die Leistungsfähigkeit und Vielseitigkeit von Neuronalen Netzen wird erkauft mit einer sehr viel aufwändigeren Parametrisierung. Eine systematische Optimierung dieser Hyperparameter wird nur durch fortgeschrittene AutoML Technologien überhaupt möglich.

Automatisierung

Es gibt Open Source Tools, wie z.B. das an der Universität Freiburg entwickelte BOHB, die mit fortgeschrittenen mathematischen Methoden und einigen Tricks den Prozess der Parameteroptimierung automatisieren und beschleunigen.

Mit einem Aufwand von rund 1-3 Tagen (etwas Vertrautheit mit der nicht ganz einfachen API vorausgesetzt) lässt sich ein solches Framework auf das Modell anwenden. Die Parametersuche selber ist ressourcenintensiv, und sollte auf einer MultiCore-GPU Machine oder einem Rechencluster laufen, da in der Regel einige Stunden oder Tage Rechenzeit erwartet werden. Die Kosten für die

erforderlichen Rechnerkapazitäten bei einem großen Cloud-Anbieter liegen i.d.R. im niedrigen dreistelligen Euro Bereich.

Einsparpotentiale durch High-Level Frameworks
Bei Einsatz eines Frameworks, das eine parallelisierte AutoML-Optimierung der Parameter sowohl des Modells als auch der Datencodierung bereits integriert, lässt sich der Programmieraufwand hierfür um ca. 60-80% reduzieren.

5) Tests
Nach Fertigstellung von Codierungs-Pipeline und Modell erfolgt in der Regel eine Phase, in der das Modell mit verschiedenen Konstellationen von Daten trainiert und getestet wird.

Automatisierungspotentiale
Tests können unterstützt werden mit Methoden zur systematischen Variation von ausgewählten Parametern aus der gesamten Prozesskette z.B. Gittersuche, dem Logging und Präsentation von Ergebnissen.

Einsparpotentiale durch High-Level Frameworks
Schnelleres Arbeiten durch einfachere Anpassungen im Code, einfachere Parametrisierung von Experimenten, sowie smartes Caching bei der Pipeline (reduzierte Wartezeiten bei wiederholt ausgeführten Preprocessing Schritten). Einsparpotentiale lassen sich mit 50% oder mehr abschätzen.

Durch Einsatz geeigneter Technologien lassen sich bei der Umsetzung von ML Projekten Effizienz-gewinne in der Größenordnung von 70% realisieren, ohne Einbußen bei der Flexibilität.

6) Deployment
Das Deployment eines trainierten Machine Learning Modells, und damit die Umsetzung der Ergebnisse einer Machbarkeitsstudie in den Livebetrieb ist oft ein überraschend aufwändiger Prozess. Der gesamte Code muss vom Labor-Stadium in produktive Qualität gehoben werden - alle zuvor genommenen Abkürzungen wollen nun mit umso größerer Aufmerksamkeit nachgeholt werden. Zusätzlich sind Themen wie das Monitoring nicht nur der korrekten Funktionsweise des Modells, sondern auch der Beschaffenheit der Eingabedaten wichtig. Alle datengetriebenen Komponenten, also viele Encoder in der Vorverarbeitungspipeline und natürlich das Modell selbst, müssen zuverlässig abgespeichert und wieder geladen werden können.

Frameworks, die das "Einpacken", die Versionierung und das Deployment von Modell und Datenvorverarbeitung unterstützen, und damit über den Funktionsumfang der gängigen Open Source Bibliotheken wie scikit-learn hinausgehen, können diese Arbeit erheblich effizienter und die Ergebnisse robuster machen. Tensorflow bietet hierfür einige Tools, die den Benutzer jedoch auf das Tensorflow-Universum festlegen.

7) Übersicht Einsparungspotentiale durch High-Level Frameworks
Der größte Aufwand, und somit auch die größten Einsparungspotentiale ergeben sich bei der Vorverarbeitung der Daten. Über den gesamten Entwicklungszyklus konnten wir bei konkreten Applikationen Einsparungspotentiale in der Größenordnung von 70% bzgl. des Codes beobachten, die Einsparung an Arbeitszeit dürfte ähnlich oder höher sein, weil der Code in höher aggregierten Umgebungen überwiegend aus high-level Kommandos bzw. Parametrisierungen in vergleichsweise fehlersicheren Umgebungen besteht.

Design choices

Parallelisierung
Für einige rechenintensive Prozesse, hier etwa insbesondere die Schema-extraktion und Datentransformation, sowie die automatisierte Hyperparameteroptimierung von Neuronalen Netzen, ist eine parallelisierte Verarbeitung sehr empfehlenswert. Die dadurch stark reduzierte "Wall-Clock Time" erlaubt auch mit großen Datenmengen weiterhin so iterativ und experimentell zu arbeiten wie mit kleineren. Dies erfordert ein Parallelisierungs-Framework, das für die interaktive Anwendung optimiert ist.

Apache Spark war dafür lange Zeit ein Mittel der Wahl: Aufbauend auf Hadoop, Kubernetes, Apache Mesos oder anderen Plattformen ermöglicht es interaktive Datenverarbeitung, Analysen und Machine Learning mit Java, Scala, Python, R und SQL. Für die inzwischen vorwiegend in Python arbeitenden Data Scientists bringt die darunter liegende JVM aber auch einige Komplexität mit sich. Gerade wenn es nur um die Parallelisierung von Rechenprozessen und nicht um SQL-Integration geht, hat sich für uns der Einsatz von Dask bewährt. Dieses Framework ist nativ in Python implementiert und bietet Low-Level Job Cues, parallelisierte Collections ("Bags") ähnlich den RDDs in Spark, und darüberhinaus auch die Verteilung von großen Numpy- und Pandas-Datenstrukturen. Für uns ist Dask daher eine deutlich bessere und unkompliziertere Lösung als Apache Spark.

Für den Einsatz in der Python Welt bewährt sich Dask als leistungsfähiges und vergleichsweise leichtgewichtiges Parallelisierungs-framework.

Machine Learning Modell / Deep Learning
Neuronale Netze (“Deep Learning”-Modelle) nehmen heute unter den ML-Modellen eine Sonderstellung ein. Obwohl sich ihre Entwicklung bis in die Mitte des letzten Jahrhunderts zurückverfolgen lässt, haben sie erst ab 2006 vor allem aufgrund gewachsener Rechnerkapazitäten eine Reihe von spektakulären Durchbrüchen erzielt und sind heute insbesondere bei sehr komplexen Problemen mit hochdimensionalen Daten State of the Art. Deep Learning ist Gegenstand intensiver Forschung und für viele Aufgaben wie semantische Textverarbeitung, Bildklassifikation, Zeitreihen oder die Prozesssteuerung wurden jeweils spezielle Architekturen entwickelt. Diese können wiederum miteinander kombiniert werden, etwa in **Multi-Channel Networks**, wo innerhalb des Netzwerks zunächst verschiedene Eingänge durch jeweils eigene Layer separat prozessiert werden, ehe die Ergebnisse dann zusammengeführt werden und mit weiteren Layern ein oder auch mehrere Outputs generiert werden.

Nachteile von Neuronalen Netzen sind vor allem die aufwändige Parametrisierung, die schwierige Interpretierbarkeit und das ressourcenintensive Training. Inzwischen konnten mit AutoML, Verfahren wie Shapley Values sowie Parallelisierung und GPUs Wege gefunden werden, um diesen Schwierigkeiten zu begegnen, und so erfreuen sich Neuronale Netze auch in kommerziellen Anwendungen wachsender Beliebtheit.

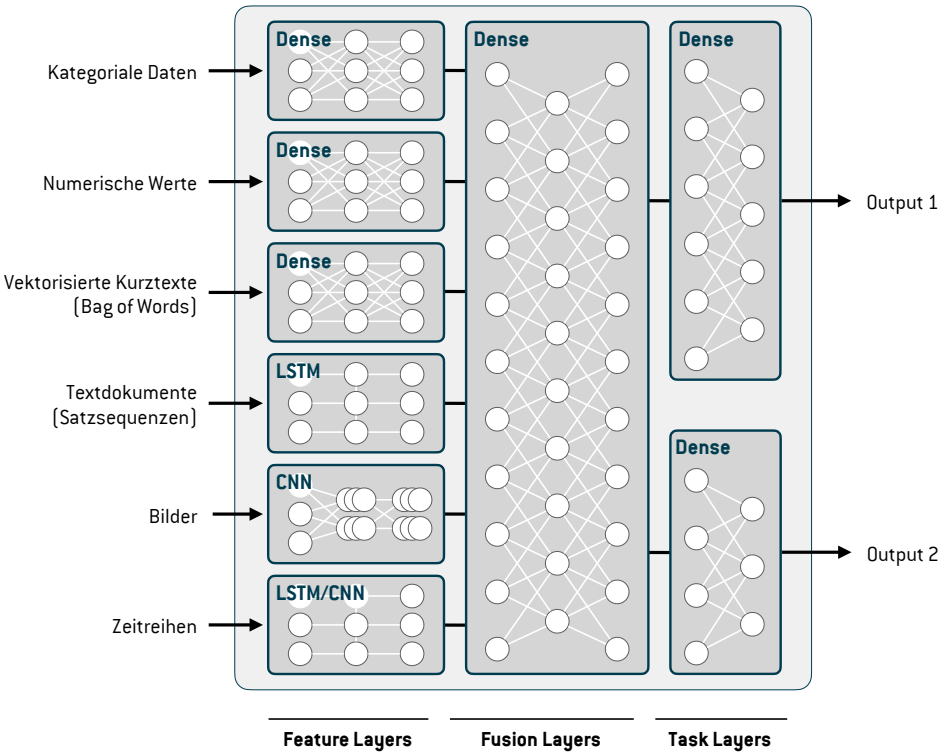

Abb. 3: Ein Multi-Channel Neural Network kann beliebige Kombinationen von Daten verschiedenen Typs prozessieren und auf einen oder mehrere Outputs abbilden.

Unser Use Case, ein Klassifikator für GDSN-Daten, wäre mit verschiedenen Machine Learning Algorithmen lösbar, wie z.B. Random Forests, Boosting Algorithmen, die sich gut für Daten gemischter Typen eignen, oder auch einer einfachen Logistic Regression. Ein Multi-Channel Neural Network ist für diese Aufgabenstellung ein relativ komplexes Modell. Ausschlaggebend für diese Auswahl war, neben der Performance, dass diese Architektur sehr elegant erweiterbar ist auf beliebige Kombinationen komplexerer Datenstrukturen, wie z.B. größere Textdokumente, Bilder, Zeitreihen, Audio oder Video.

Zudem kann in vielen Fällen von Nutzen sein, dass sich Neuronale Netze zwanglos auf Multi-Label Klassifikatoren erweitern lassen, die ohne Performance-Einbußen viele sich nicht ausschließende Klassenzugehörigkeiten gleichzeitig lernen können.

Darüberhinaus lassen sich aus diesen Modellen mit wenigen weiteren Handgriffen leistungsfähige unsupervised Modelle wie AutoEncoder sowie darauf basierende fortgeschrittene Clustering Modelle bauen, die Ähnlichkeiten zwischen Artikeln quantifizieren und Anomalien entdecken können, etwa zur Unterstützung bei der Fehlersuche in den Daten (siehe Use Cases), oder für Unsupervised Pretraining in Fällen, wo nur wenige gelabelte Trainingsdaten zur Verfügung stehen.

Ein Multi-Channel Neural Network kann Input Daten verschiedenen Typs mit jeweils geeigneten Substrukturen verarbeiten.

Fazit

Unternehmen, sei es im Retail-Sektor oder anderen Branchen, die Machine Learning in ihr Geschäft integrieren wollen, steht hierfür eine große Auswahl an Werkzeugen und Strategien zur Verfügung. Die Spannbreite reicht hierbei von relativ aufwändiger eigener Programmierung unter Verwendung jeweils passender Open Source Tools bis zum Einsatz voll integrierter und automatisierter Lösungen wie DataRobot oder Dataiku.

Für Unternehmen gilt es die Balance zu finden zwischen der vollen Flexibilität der Lösungsentwicklung from scratch und dem Automatisierungspotential höher aggregierter Umgebungen. Je größer ein Unternehmen und je strategisch bedeutender die Integration von KI ist, umso eher wird es sich für den Aufbau eines maßgeschneiderten Ökosystems entscheiden und auf Lösungen setzen, die einerseits möglichst viel Flexibilität bewahren, und doch technologische Potentiale zur Automatisierung und Vereinfachung konsequent nutzen.

Je größer ein Unternehmen und je strategisch bedeutender die Integration von KI ist, umso eher wird es sich für den Aufbau eines proprietären ML Ökosystems entscheiden.

Über Learning Machines

Learning Machines ist ein Startup, das aus dem Umfeld der Universität Freiburg gegründet wurde, wo KI mit mehreren Lehrstühlen stark vertreten ist, u.A. mit einem Forschungsschwerpunkt auf Automated Machine Learning.

Gestützt auf die Erfahrung aus mehr als 20 kommerziellen Data Science Projekten und gefördert über ein BMWi Projekt hat das Unternehmen die Learning Machines Engine entwickelt. Diese AutoML Bibliothek von Data Scientists für Data Scientists aggregiert auf Basis von state-of-the-art Technologien viele der hier besprochenen Methoden, einschließlich Neuronaler Netze und der effizienten Aufbereitung von GDSN-Rohdaten.

Learning Machines bietet Beratung rund um KI und Machine Learning sowie die Umsetzung von maßgeschneiderten ML Lösungen auf Basis unserer Engine.

Kontakt

Learning Machines GmbH
+49 761 7043799
info@learning-machines.de
Gerberau 9a
79098 Freiburg i. Br.