Labeling mit Active Learning

Sparsam etikettiert

Maximilian Blanck, Matthias Richter

Das Labeln großer Datensätze für das überwachte Training von ML-Modellen ist aufwendig. Active Learning findet selbstständig die Datenpunkte, bei denen die Mühe lohnt, und spart so viel Arbeit. Für Python steht dafür die Bibliothek modAL zur Verfügung.

Drei Dinge sind für überwachtes maschinelles Lernen essenziell: Daten, Algorithmen und Labels. An den ersten beiden herrscht kein Mangel. Bei Labels – also den Kategorien, in die Trainingsdaten eingeordnet werden – sieht es anders aus. Während Beispieldatensätze aus dem Netz oft sorgfältig gelabelt sind, sind bei Daten aus der Praxis unvollständige oder mangelhafte Labels eher die Regel als die Ausnahme. Soll ein Algorithmus etwa im echten Leben Kreditkartenbetrug erkennen, hat selten mehr als eine Handvoll Daten die richtigen Labels.

Twitter-Posts labeln

In den meisten Fällen muss man seine Daten manuell labeln. Das ist zeitraubend, kostspielig, fehleranfällig und kann zu suboptimalen Modellen führen. Ganz besonders ärgerlich wird es dann, wenn viele der Labels für den Lernalgorithmus gar nicht wichtig sind, weil er entweder schon genug ähnliche Datenpunkte gesehen hat oder es sich um Ausreißer handelt. Hier hilft Active Learning: Der Algorithmus sucht aktiv nach interessanten Datenpunkten und legt sie einem Menschen zum Labeln vor. Das minimiert einerseits den Aufwand und führt andererseits zu robusteren Modellen.

Ein Beispiel soll zeigen, wie Active Learning im Detail funktioniert, und die Frage beantworten, in welchem Maß es ohne Qualitätsverlust Trainingsdaten einspart. Als Grundlage dient ein bereits erfolgreich eingesetztes System zur Sentimentanalyse von Tweets, das Texte automatisch in positive und negative Aussagen einteilt. Es wurde ursprünglich mit 40 000 gelabelten Tweets trainiert – 40 Prozent mit negativer und 60 Prozent mit positiver Aussage. Als Features nutzt das System das Verfahren Term Frequency – Inverse Document Frequency (Tf-idf), das Wörter anhand ihres Vorkommens in einer Menge von Dokumenten gewichtet. Anschließend klassifiziert es die Tweets mit einer Support Vector Machine (SVM).

Um nun Active Learning zu simulieren, wird das System komplett neu trainiert. Dabei wird Active Learning mit Random Sampling verglichen, dem zufälligen Auswählen des nächsten Datenpunktes zum Labeln.

Dem Vergleich liegt der typische Ansatz zugrunde, Test- und Trainingsdaten zu trennen. Ausgehend von einer kleinen Menge an Lerndaten und Labels trainiert man ein Modell und evaluiert es anhand einer separaten Menge an Testdaten. Dann erweitert man den Trainingsdatensatz und trainiert und evaluiert das Modell erneut. Die Erweiterung des Lerndatensatzes erfolgt jeweils mit Active Learning, das Daten aktiv auswählt, oder zufällig mit Random Sampling.

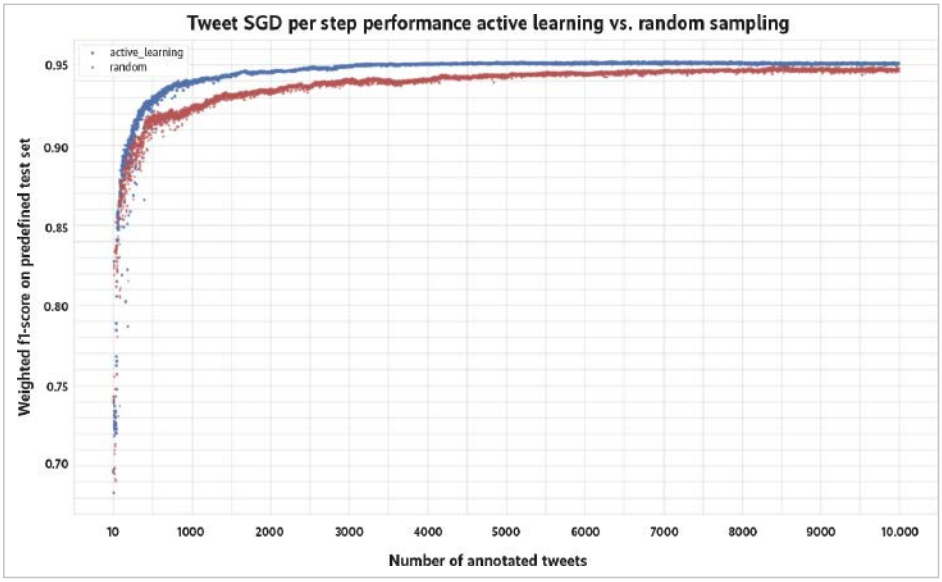
Zu Beginn der Simulation teilt man die 40 000 Tweets in zwei Pools zu je 20 000 Tweets zufällig auf und zieht bei beiden Ansätzen initial 10 zufällige Trainingsdaten. Danach werden je nach Ansatz aus dem Pool die nächsten Datenpunkte aktiv oder zufällig dem Trainingsdatensatz hinzugefügt. Als Gütemaß dient der F1-Score, also das geometrische Mittel aus Genauigkeit (dem Anteil der positiven Tweets an den als positiv klassifizierten Tweets) und Trefferquote (dem Anteil an gefundenen positiven Tweets). Abbildung 1 stellt die Anzahl der Trainingsdaten auf der x-Achse und die Klassifikationsgüte anhand des F1-Scores auf der y-Achse dar.

Jeder Punkt in der Abbildung steht für eine Runde in der Simulation: Rot für Random Sampling, Blau für Active Learning. Zu Beginn der Simulation, wenn nur wenige Trainingsdaten verfügbar sind, unterscheiden sich beide Strategien kaum. Ab 250 gelabelten Tweets ist Active Learning

iX-TRACT

- Active Learning findet Datenpunkte, bei denen ein Label zweckmäßig ist.
- Damit werden weniger gelabelte Daten gebraucht, um gute ML-Modelle zu trainieren, und der Labelling-Aufwand sinkt.
- Der Active-Learning-Zyklus lässt sich mit der Python-Bibliothek modAL unkompliziert umsetzen.

jedoch dem Random Sampling überlegen. Bei 3000 bis 5000 gelabelten Tweets ist das System mit Active Learning bereits so gut trainiert, dass es sich kaum noch verbessert. Random Sampling erreicht hingegen selbst mit 10 000 gelabelten Tweets nicht das Qualitätsniveau wie Active Learning. In diesem Beispiel hätte Active Learning also nicht einmal die Hälfte der Daten labeln müssen und das gelernte Modell wäre trotzdem besser als mit Random Sampling.



Bei der Simulation von Active Learning im Vergleich zu Random Sampling ist Ersteres bei wenigen gelabelten Datenpunkten klar überlegen. Selbst bei 10 000 Datenpunkten erreicht Random Sampling nicht ganz dieselbe Qualität (Abb. 1).

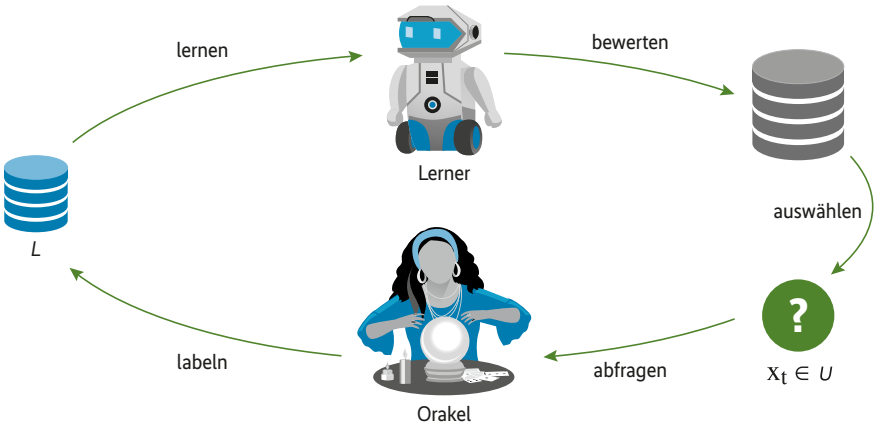
So funktioniert Active Learning

Das zuvor beschriebene Szenario ist als Pool-based Active Learning bekannt (siehe Abbildung 2). Dem Lerner steht ein großer Pool aus ungelabelten Daten *U* sowie ein deutlich kleinerer Pool aus gelabelten Trainingsdaten *L* zur Verfügung. Der Lerner lernt ein Modell anhand der Trainingsdaten *L* und nutzt dieses Modell, um alle noch nicht gelabelten Daten *U* anhand ihres Nutzens für den Lernfortschritt zu bewerten. Das Orakel, eine Maschine oder ein Mensch, erhält dann die nützlichen ungelabelten Daten. Es labelt die Daten und übergibt sie dem Lerner, der damit ein neues Modell trainiert. Dieser Zyklus wiederholt sich, bis das Modell gut genug oder das für das Labeling zur Verfügung stehende Budget aufgebraucht ist.

Pool-based Active Learning ist vor allem dann interessant, wenn ein neues Modell gelernt werden soll. Wenn sich das Modell dagegen bereits im Einsatz befindet und sich verbessern soll, bietet sich Selective Sampling an (siehe Abbildung 3). Hier bekommt der Lerner Datenpunkte zum Klassifizieren vorgelegt und kann nun entscheiden, ob er diese an das Orakel weiterreichen muss oder die Klasse selbst bestimmt.

Zusätzlich zu den vorgestellten Szenarien nennt das Active-Learning-Standard-

werk von Burr Settles [2] noch ein drittes Szenario: Query Synthesis. Hier erzeugt der Lerner synthetische Datenpunkte, die er dem Orakel vorlegt. Dieses Szenario ist aber meist nur von akademischem Interesse. Grund dafür ist, dass man ein generatives Modell benötigt, das beschreibt, wie man einen Datensatz im Rahmen eines Wahrscheinlichkeitsmodells erzeugt. Zudem könnte man mit dieser Methode auch Daten erfinden, die selbst das Orakel nicht annotierten kann.



Das Orakel labelt die Daten und gibt sie an den Lerner weiter (Abb. 2).

Daten und ihr Nutzen

In all diesen Szenarien benötigt man ein Bewertungsmaß, das die Nützlichkeit eines Datenpunktes für den Trainingsfortschritt bestimmt. Ein effektiver Ansatz hierfür ist das Uncertainty Sampling. Hierbei muss das Machine-Learning-Modell alle Datenpunkte im Pooldatensatz bewerten, indem es beispielsweise eine Klassenwahrscheinlichkeit für jeden Datenpunkt berechnet. Die Grundidee ist einleuchtend: Je unsicherer das Modell bei Vorhersage ist, desto wertvoller ist es für den Lerner, das Label eines Datenpunktes zu kennen.

Die Messung der Unsicherheit der Vorhersage hängt vom Modell ab. Für probabilistische Modelle, die für jede Klasse eine Wahrscheinlichkeit berechnen, bietet sich der Least Confidence Score an. Hierbei ist ein Datenpunkt umso nützlicher, je geringer die vorhergesagte Wahrscheinlichkeit der sichersten Klasse ist. Der nützlichste Datenpunkt ist der, bei dem das Modell alle Klassen für gleich wahrscheinlich hält. Dieses Maß ist bei vielen nicht probabilistischen Modellen einsetzbar. Meist codiert die Entscheidungsfunktion selbst die Unsicherheit: Die SVM ist beispielsweise umso unsicherer, je näher die Entscheidungsfunktion an 0 reicht.

Es ist nicht schwer, Pool-based Active Learning mit Uncertainty Sampling selbst zu implementieren (Listing 1). Es gibt aber auch gut gepflegte Bibliotheken wie modAL, die viele gängige Methoden mitbringen: Sie baut auf scikit-learn auf und lässt sich einfach in bestehende scikit-learn-Projekte einbinden. Mit entsprechenden Wrappern kann man modAL aber auch mit anderen Ökosystemen verwenden. TensorFlow besitzt einen solchen Wrapper: `tf.keras.wrappers.scikit_learn`. PyTorch-Modelle kann man mittels skorch mit modAL benutzen (siehe ix.de/zagz).

Praktische Umsetzung

Da schon das Ausgangsprojekt im Beispiel mit dem scikit-learn-Stack realisiert wurde, dient dieser auch für die Simulation. Listing 2 zeigt die Einteilung der gelabelten Tweets in Pool- und Testdaten und das Ziehen von zehn gelabelten Tweets für die initialen Trainingsdaten. Der Rest bildet den Pool ungelabelter Daten.

Es ist wichtig, dass der Trainingsdatensatz sowohl Positiv- als auch Negativbeispiele enthält, da die SVM schon initial beide Klassen kennen muss. In diesem Beispiel verzichtet man aber beim zweiten Splitten auf Stratifizierung, also das Gleichverteilen der jeweiligen Klassen auf zwei Datensätze. Dadurch soll die Simulation möglichst nahe an die Realität herankommen. Vor dem Labeling weiß man schließlich nicht, ob ein Tweet positiv oder negativ konnotiert ist.

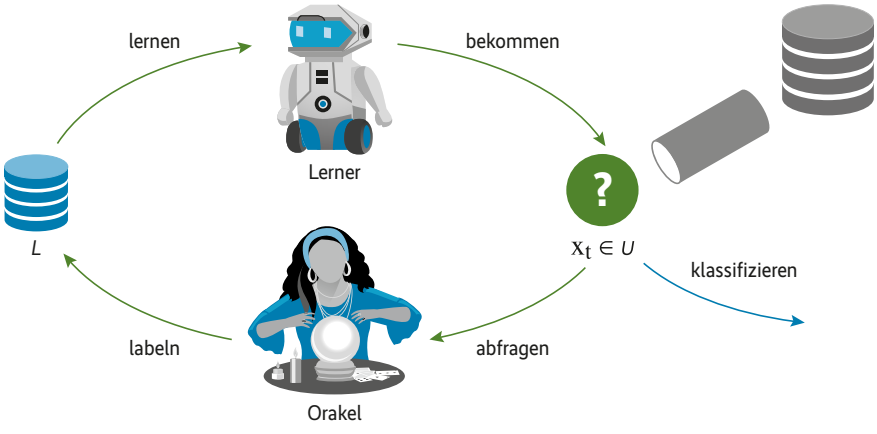
Der eigentliche Active-Learning-Zyklus lässt sich bequem mit modAL umsetzen. Die Bibliothek benutzt dafür die Klasse `ActiveLearner`, in der die wesentlichen Parameter festgelegt sind. In diesem Beispiel entspricht das dem zugrunde liegenden Tweet-Klassifikator, Uncertainty Sampling als Nützlichkeitsmaß (Querystrategie), sowie dem initialen Lerndatensatz, der das Modell bei der Instanziierung von `ActiveLearner` trainiert (siehe Listing 3).

Im weiteren Verlauf verwaltet modAL die Trainingsdaten und erweitert sie mit neuen Labels vom Orakel. Der gesamte Active-Learning-Zyklus lässt sich bereits mit wenigen Codezeilen ausführen (siehe Listing 4).

Der entscheidende Schritt erfolgt mit dem Aufruf `query(X_pool)`. Er bewertet alle ungelabelten Datenpunkte in `X_pool` mittels Querystrategie (hier Uncertainty Sampling). Der Rückgabewert ist der Index des Datenpunktes, bei dessen Vorhersage das Modell am unsichersten war.

Der nächste Aufruf `teach(data, label)` erweitert den Trainingsdatensatz um den Datenpunkt und dessen Label `y_pool[query_idx]` und trainiert das Machine-Learning-Modell auf dem aktualisierten Datensatz neu. In einem realen Szenario müsste hier ein Orakel das Label erzeugen.

Die Schritte wiederholt man in einer Schleife, bis ein bestimmtes Abbruchkriterium eintritt, beispielsweise bis die Modellgüte ausreicht oder genügend Daten gelabelt sind.



Beim Selective Sampling entscheidet der Lerner, ob er dem Orakel Daten vorlegt (Abb. 3).

Fazit

Beim praktischen Einsatz von Active Learning haben sich einige Vorgehensweisen bewährt. Wenn möglich sollte man mit einer Simulation beginnen, wie sie hier beschrieben ist. Falls dafür nicht genug gelabelte Daten verfügbar sind, kann man auch Daten aus einer verwandten Domäne benutzen. Die Test- und Validierungsdaten sollte man vor dem Active Learning aus-

Listing 1: Implementierung von Pool-based Active Learning mit Uncertainty Sampling

```
def pool_based_round(clf, X_train, y_train, X_pool, oracle):
  # Klassifikator trainieren
  clf.fit(X_train, y_train)
  # Bestimmung des nützlichsten Samples
  idx = uncertainty_sampling(clf, X_pool)

  most_useful_data = X_pool[idx]
  # Label erfragen
  label = oracle(most_useful_data)
  # Trainingsdaten erweitern
  X_train = np.concatenate((X_train, most_useful_data.reshape(1, -1)))
  y_train = np.concatenate((y_train, oracle(most_useful_data)))
  # Sample aus ungelabelten Daten entfernen
  X_pool = np.r_[X_pool[:idx], X_pool[idx+1:]]
  return clf, X_train, y_train, X_pool

def uncertainty_sampling(clf, X_pool):
# Schätzung der Klassenwahrscheinlichkeiten für jedes Sample in X_pool
  proba = model.predict_proba(X_pool)
  # Wahrscheinlichkeitsschätzung der sichersten Klasse für jedes Sample
  highest_proba = proba.max(axis=1)
  # Index des unsichersten Samples
  return highest_proba.argmin()
```

Listing 2: Vorbereitung der Daten

```
from sklearn.model_selection import train_test_split

   pool_tweets, test_tweets, pool_labels, test_labels = \
       train_test_split(tweets, labels, test_size=20000, stratify=labels, random_state=1)

   X_pool, X_train, pool_labels, y_pool = \
       train_test_split(pool_tweets, pool_labels, test_size=10)
```

wählen, da die Qualität immer anhand eines unabhängigen Datensatzes gemessen werden sollte.

Wie bei allen Machine-Learning-Projekten sollte man möglichst einfach beginnen, also mit Uncertainty Sampling. Daneben sollte man aber auch andere Nützlichkeitsmaße in Betracht ziehen. Außerdem ist die Integration von Domänenwissen in den Auswahlprozess wichtig. Libraries wie modAL bringen diese Methoden in der Regel mit.

Zuletzt muss man auf die Wahl des Machine-Learning-Modells achten. Weil das Modell die Auswahl der Trainingsdaten beeinflusst, können andere Modelle, die man mit demselben Datensatz trainiert, schlechter abschneiden.

Trotz Einschränkungen lohnt sich der Einsatz von Active Learning – besonders dann, wenn das Labeling teuer ist. In diesem Beispiel hätten weniger als 5000 statt der 40 000 vorhandenen Labels ausgereicht. Zudem setzt Active Learning am Anfang der Machine-Learning-Kette an, weshalb es sich relativ einfach in bestehende Projekte integrieren lässt. (ulw@ix.de)

Quellen

[1] Links zu Methoden des Active Learning, Dokumentationen von modAL, dem TensorFlow-Wrapper und skorch: ix.de/zagz

[2] Burr Settles; Active Learning; Morgan & Claypool 2012

[3] David Foster; Generatives Deep Learning; O'Reilly 2020

Maximilian Blanck

arbeitet als Data Scientist für inovex. Er exploriert Daten, entwickelt Machine-Learning-Modelle und befasst sich mit den Themen Natural Language Processing, Deep Learning und Statistik.

Matthias Richter

ist Machine Learning Engineer bei inovex. Er beschäftigt sich mit Machine-Learning-Projekten von Datenerfassung und Bereinigung über Modelltraining und Evaluation bis hin zu Deployment.

Listing 3: Instanziierung der Active-Learner-Klasse

```
from modAL.models import ActiveLearner
from modAL.uncertainty import uncertainty_sampling

learner_al = ActiveLearner(

    estimator=sgd_tweet_classifier,
    query_strategy=uncertainty_sampling,
    X_training=X_train, y_training=y_train
)
```

Listing 4: Active-Learning-Zyklus ausführen

```
# Man kann ein beliebiges Abbruchkriterium benutzen
while not done:

        # Den nützlichsten ungelabelten Tweet finden
        query_idx, _ = learner_al.query(X_pool)

        # Das Orakel ist hier mit y_pool[query_idx] implementiert
        learner_al.teach(X_pool[query_idx], y_pool[query_idx])
```