Sentiment-Analyse; Klassifikation mit dem Perceptron-Algorithmus

Benjamin Roth

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilian-Universität München
beroth@cis.uni-muenchen.de

Sentiment-Analyse

Sentiment-Analyse

- Sentiment analysis, Opinion Mining
- Erkennen von Meinungen, Werturteilen, Bewertungen, Positiven/Negativen Emotionen
- Für einen gesamten Text oder speziell in Bezug auf eine bestimmte Entität (Produkt, Thema, Person, Ereignis ...)
- Anwendung z.B.
  - in Unternehmen
  - in der Sozialforschung

Sentiment-Analyse: Beispiel

Posted by: XYZ, Date: September 10, 2011
I bought an ABC Camera six months ago. I simply love it. The picture quality is amazing and the battery life is also long. However, my wife thinks it is too heavy for her.

- **Entity**: Meinung über ABC Camera
- **Aspects**: Bildqualität, Batteriedauer, Gewicht der ABC Camera
- **Opinion Holder**: XYZ, Frau von XYZ
- **Sentiment**: Bewertung der jeweiligen Aspekte durch Opinion Holder
- **Date**: September 10, 2011

Vereinfachte Problemstellung

- Sentiment-Klassifikation auf Dokumentebene.
- Problemstellung: Gegeben ein Opinion-Dokument, bestimme das generelle Sentiment des Opinion Holders gegenüber der Entität.
- Annahmen:
  - Das Dokument drückt genau ein Sentiment aus.
  - Der Opinion Holder ...
  - Die Entität ...
  - Die Zeit ...
  - Die Aspekte ...
  - ... sind bekannt oder irrelevant.
- z.B. Gegeben eine Filmbewertung (z.B. von IMDB) ist diese Bewertung **positive** oder **negativ**
  - Wie könnte eine 10-Punkte Bewertung in diese Kategorien überführt werden?
  - Wie könnte eine neutrale Kategorie mitvorhergesagt werden?

Beispiel (IMDB)

The sopranos was probably the last best show to air in the 90's. its sad that its over, its was the best show on HBO if not on TV, not everything was spelled out for you throughout you had to think, it was brilliant. the cast was excellent. Tony (James Gandolphini) is a great actor and played his character excellent, as well as the others.

I am not one of those people who just go online after I see a movie and decide to call it the worst movie ever made. If you doubt me, please look at my other reviews. However, for the first time ever, I have seen a movie so horrible that I wanted to write about how bad it was before it was even over.

Lösungsvorschläge

- Algortihmen?
- Ressourcen?
- Merkmale?

Lösungsvorschlag: Regelbasiert

- Benutze eine manuell erstellte Liste mit Polaritätsmarkierern
  - Wörter (meist Adjektive), die eindeutig einer Polarität zugeordnet sind.
  - Es existieren mehrere solcher Wortlisten (*polarity lexicons, sentiment lexicons*):
    Bing Liu's Opinion Lexicon, MPQA Subjectivity Lexicon,
    SentiWordNet, Harvard General Inquirer, LIWC ...
  - Übersicht:
    `http://sentiment.christopherpotts.net/lexicons.html`
- Solche Listen können auch halbautomatisch erstellt werden
  - Starte mit einer kleinen Anzahl von Wörtern für positive und negative Klasse
  - Erweitere jeweils mit den ähnlichsten Wörtern (z.B. aus Wordnet oder durch Co-Okkurenzen)
- Zähle nach, ob mehr positive oder negative Markierer in einem Text vorkommen.

Lösungsvorschlag: Maschinelles Lernen

- Verwende keine vorgegeben Wortliste, sondern lerne Merkmale aus annotierten Daten.
- Vorteile:
  - Benötigt keine erstellte Liste.
  - Findet auch Merkmale, an die ein Annotator nicht denken würde.
  - Feinere Gewichtung der Merkmale.
  - Viele verschiedene Repräsentationen und Klasifikatoren möglich (z.B. BOW+Naive Bayes, Neuronale Netze, ...)
  - Kann mit regelbasiertem Ansatz verbunden werden (z.B. durch Merkmal: Anzahl der Markierer)
  - ...
- Nachteile:
  - Benötigt viele annotierte Daten.
  - Overfitting, Domänenabhängigkeit.
  - ...

Plan für diese Woche

- Einfache Sentiment-Klassifikation auf Dokumenten-Ebene
- Klassifikator: Perzeptron
- Merkmale: Bag-Of-Words

Perzeptron-Algorithmus

Perzeptron-Algorithmus

- Der Perzeptron-Algorithmus fällt in die Gruppe der **diskriminativen** Modelle
- optimiert die Qualität der Vorhersage.
- *"Gegeben die Merkmale, was ist die korrekte Klasse?"*

Perzeptron Algorithmus: Idee

- Für jedes mögliche Merkmal (z.B. Wort) wird ein Gewicht gelernt (Zahl, die positiv oder negativ sein kann).
- Vorhersage der Klasse für eine Instanz:
  - Für jedes Merkmal wird der Wert des Merkmals (z.B. Vorkommen im Dokument) mit dem jeweiligen Merkmalsgewicht multipliziert, die Ergebnisse werden aufsumiert.
  - Ist der resultierende Wert größer als 0, wird die positive Klasse vorhergesagt (POS), ansonsten die negative Klasse (NEG)
- Der Perzeptron-Algorithmus passt die Merkmalsgewichte iterativ so an, dass Fehklassifikationen möglichst minimiert werden.

Darstellung der Instanzen als Vektoren

- Jede Instanz i kann als Vektor $x^{(i)}$ dargestellt werden (wie auch schon bei der TF-IDF-Berechnung).
- Die Anzahl der Komponenten des Vektors entspricht der Größe des Merkmalsraums (z.B. des Vokabulars).
- Bei einem Vokabular der Größe n ist $x^{(i)} \in \mathbb{R}^n$
- Jede Komponente des Vektors entspricht einem Wort.

Beispiel-Instanz...

...als Vektor für das Vokabular `[movie, entertain, highly, waste, time]`:

movie
time
waste

$$x^{(i)} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

- Implementierung: Da die meisten Einträge 0 sind, wählen wir eine effiziente Darstellung mit Dictionaries (Wort $\Rightarrow$ Anzahl)

Vektormultiplikation

- Wiederholung Vektormultiplikation (Whiteboard)

Vektormultiplikation mit Gewichtsvektor

- Gewichtsvektor

$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix}$$

- Vorhersagewert (Score) für Instanz i:

$$x^{(i)T} w = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} = w_1 + w_4 + w_5$$

- Implementierung: Auch für den Gewichtsvektor wählen wir eine Darstellung mit Dictionaries (Wort $\Rightarrow$ Gewicht)

Vorhersage

- Entscheidungsregel:
  - $x^{(i)T} w > 0 \Rightarrow$ POS
  - $x^{(i)T} w <= 0 \Rightarrow$ NEG
- Beispiel:

movie
time
waste

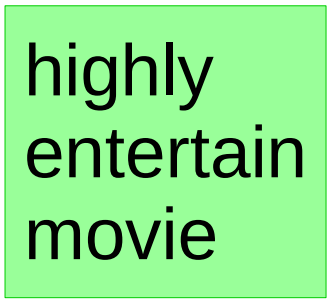
highly
entertain
movie

- $x^{(1)T} w = w_{\text{movie}} + w_{\text{waste}} + w_{\text{time}}$
- $x^{(2)T} w = w_{\text{movie}} + w_{\text{entertain}} + w_{\text{highly}}$

- Welche Gewichte würden zu einer Fehlklassifikation beider Instanzen führen (beliebiges Beispiel)?
- Wie müssten diese Gewichte korrigiert werden, damit die Instanzen richtig klassifiziert werden?

Vorhersage

- Entscheidungsregel:
  - $x^{(i)T} w > 0 \Rightarrow$ POS
  - $x^{(i)T} w \leq 0 \Rightarrow$ NEG
- Beispiel:



  - $x^{(1)T} w = w_{\text{movie}} + w_{\text{waste}} + w_{\text{time}}$
  - $x^{(2)T} w = w_{\text{movie}} + w_{\text{entertain}} + w_{\text{highly}}$

- Gewichte, die zu einer Fehlklassifikation beider Instanzen führen: *z.B.* $w_{\text{movie}} = 1.0; w_{\text{waste}} = -2.0; w_{\text{time}} = 1.5; w_{\text{entertain}} = -2.0; w_{\text{highly}} = 0.5$
- Korrektur der Gewichte:
  - w_{waste} *und/oder* w_{time} *muss verkleinert werden*
  - $w_{entertain}$ *und/oder* w_{highly} *muss vergrößert werden.*
  - w_{movie}*: Unentschieden für beide Instanzen.*
  - *alle anderen Gewichte neutral.*

Anpassung der Gewichte (Perzeptron-Update)

- Idee:
  - Falls richtige Vorhersage für Trainings-Instanz: mache nichts.
  - Sonst: Erhöhe/verringere Gewichte für jede Instanz so, dass der Score sich in die richtige Richtung verändert.
    Achtung: in unserem Beispiel sind nur positive Merkmalswerte vorgekommen, der Algorithmus soll aber auch mit negativen funktionieren.
  - Gewichte für Merkmale, die besonders häufig mit einer der beiden Klassen vorkommen erhalten so viele Anpassungen in die jeweilige Richtung.
  - Die Gewichte für Merkmale, die in beiden Klassen ungefähr gleichhäufig vorkommen (z.B. *for*) werden mal in die eine und mal in die andere Richtung verändert.

Perzeptron-Update für eine Instanz

- Wenn das Label übereinstimmt, kein Update der Gewichte.
- Ansonsten:
  - Wenn Vorhersage `POS` und wahres Label `NEG`: $error = 1$
    ($\Leftrightarrow$ verringern der Gewichte in Abhängigkeit der Merkmalsausprägung)
  - Wenn Vorhersage `NEG` und wahres Label `POS`: $error = -1$
    ($\Leftrightarrow$ erhöhen der Gewichte in Abhängigkeit der Merkmalsausprägung)
  - Update für Gewicht w_j (und Merkmalsvorkommen $x_j^{(i)}$)

    $w_j \leftarrow w_j - error \cdot x_j^{(i)}$

Perzeptron-Algorithmus

- **Eingabe:**
  - Merkmalsvektoren (instances) $\{x^{(1)}, x^{(2)}, ...x^{(n)}\}$
  - Labels $\{y^{(1)}, y^{(2)}, ...y^{(n)}\}$
- **Ausgabe:** Merkmalsgewichtsvektor w

```
1:  procedure PerceptronWeights(instances, labels)
2:      w = 0
3:      repeat
4:          for i = 1 to n do
5:              if x^(i)T w > 0 then
6:                  prediction = POS
7:              else prediction = NEG
8:              if prediction == POS and y^(i) == NEG then
9:                  error = 1
10:             else if prediction == NEG and y^(i) == POS then
11:                 error = −1
12:             else error = 0
13:             w = w − error · x
14:     until stopping criterion
15:     return w
```

Hinweise

- Mögliche Abbruchkriterien:
  - Vorgegebene Anzahl an Iterationen erreicht
  - Perfekte Klassifikation auf den Trainingsdaten
  - Keine Fehlerreduktion auf den Entwicklungsdaten
    $\Rightarrow$ Man nimmt dann die Gewichte aus der letzten Iteration.
- In unserem Fall waren die Merkmalswerte ganze Zahlen (Wortvorkommen), der Perzeptron-Algorithmus funktioniert aber auch mit nicht-ganzzahligen und negativen Werten.
- Der Perzeptron-Algorithmus konvergiert zu der perfekten Klassifikation der Trainingsdaten, wenn diese linear trennbar sind.
  - *linear trennbar*: es gibt eine Gewichtskombination mit perfekter Klassifikation
  - nicht alle Trainingsdaten sind trennbar,
  - perfekte Klassifikation auf Trainingsdaten generalisiert of schlecht

Hyper-Parameter für den Perzeptron-Algorithmus

- Hyper-Parameter: Parameter, die dem Algorithmus vorgegeben werden (=die dieser nicht automatisch lernt) ...
- ... werden auf den Entwicklungsdaten ausgewählt
- Welche Hyper-Parameter für Perzeptron?

Hyper-Parameter für den Perzeptron-Algorithmus

- Welche Merkmale, Anzahl der Merkmale (z.B. 1000 oder 10000 häufigste Wörter)
- Anzahl der Trainings-Iterationen
  - Vergleiche nach jeder Iteration die Accuracy auf den Entwicklungsdaten
  - Wähle Gewichtswerte der Iteration mit bester Entwicklungs-Accuracy
  - *"Early-stopping with patience n"*: Breche Training ab, falls nach $n + 1$ Iterationen keine Verbesserung auf den Entwicklungsdaten
- Schrittweite/Größe der Updates kein relevanter Hyper-Parameter (wenn Schwellwert für Entscheidungskriterium$= 0$)

Implementierung

Repräsentieren von Daten

- Eine einzelne Instanz:

```
class DataInstance:
    def __init__(self, feature_counts, label):
        self.feature_counts = feature_counts
        self.label = label
```

- Ein Datensatz:

```
class Dataset:
    def __init__(self, instance_list, feature_set = None):
        self.instance_list = instance_list
        if feature_set:
            self.feature_set = feature_set
        else:
            self.feature_set = \
              set.union(*[set(inst.feature_counts.keys()) for \
                          inst in instance_list])
```

- Korrespondenz zu TextDocument/DocumentCollection?

Perzeptron Klassifikator: Konstruktoren

- Die einzigen Parameter des Perzeptron-Klassifikators sind die Merkmalsgewichte. $\Rightarrow$ Dictionary.
- Klassifikator für ein `Dataset`: jedes Merkmal wird mit 0 inititalisiert. $\Rightarrow$ Das Modell muss noch trainiert werden.
- Laden eines schon trainierterten Klassifikators aus einer JSON-Datei.

```
class PerceptronClassifier:
    def __init__(self, weights):
        # string to int
        self.weights = weights
    @classmethod
    def for_dataset(cls, dataset):
        """ Create classifier for features in dataset.
        (hw05_perceptron.data.Dataset) """
```

Perzeptron Klassifikator: Übersicht

```
class PerceptronClassifier:
    def prediction(self, counts):
        # ...
    def update(self, instance):
        # ...
    def training_iteration(self, dataset):
        # ...
    def train(self, training_set, dev_set, iterations):
        # ...
    def test_accuracy(self, dataset):
        # ...
    def features_for_class(self, is_positive_class, topn=10):
        # ...
    def copy(self):
        # ...
    def save(self, filename):
        # ...
```

prediction und update

- $\Rightarrow$ Übungsaufgabe
- Entscheidungsregel für Vorhersage?
- Update für eine Instanz?

`prediction(counts)` und `update(instance)`

- $\Rightarrow$ Übungsaufgabe
- Entscheidungsregel für Vorhersage?
  Vektorprodukt aus Merkmalen und Merkmalsgewichten > 0*?*
- Update für eine Instanz?
  - *Vorhersage machen.*
  - *Vorhersage inkorrekt?*
  - *Je nach Fehler, Gewichte nach oben oder unten korrigieren.*

training_iteration(dataset)

- Ein Trainingszyklus auf den Trainingsdaten.
- Für jede Instanz wird Update ausgeführt (falls Fehlklassifikation).
- Vor jeder Iteration sollten die Daten neu gemischt werden.

```
def training_iteration(self, dataset):
    dataset.shuffle()
    for instance in dataset.instance_list:
        self.update(instance)
```

- (Optional kann noch die Anzahl der tatsächlichen Updates zurückgegeben werden.)

training_iteration(dataset)

- Ein Trainingszyklus auf den Trainingsdaten.
- Für jede Instanz wird Update ausgeführt (falls Fehlklassifikation).
- Vor jeder Iteration sollten die Daten neu gemischt werden.

```
def training_iteration(self, dataset):
    dataset.shuffle()
    for instance in dataset.instance_list:
        self.update(instance)
```

- (Optional kann noch die Anzahl der tatsächlichen Updates zurückgegeben werden.)

Training

- Mehrere Trainingsiterationen
- Die Accuracy wird jeweils auf den Entwicklungsdaten verglichen
- Der Klassifikator verwendet am Ende die Gewichte mit den besten Ergebnissen

```
def train(self, training_set, dev_set, iterations):
    best_dev_accuracy = 0.0
    best_weights = self.weights
    for i in range(iterations):
        errors = self.training_iteration(training_set)
        train_accuracy = self.test_accuracy(training_set)
        development_accuracy = self.test_accuracy(dev_set)
        if development_accuracy > best_dev_accuracy:
            best_dev_accuracy = development_accuracy
            best_weights = self.weights.copy()
    self.weights = best_weights
    return best_dev_accuracy
```

JSON

- Was ist JSON?
  - Kompaktes Datenformat in einer einfach lesbaren Textform
  - Übertragung von strukturierten Daten
  - Serialisierung (Speichern von Objekten/strukturierten Daten)
  - valides JavaScript (*"JavaScript Object Notation"*)
- Beispiel:

```
{
  "Herausgeber": "Xema",
  "Nummer": "1234-5678-9012-3456",
  "Deckung": 2e+6,
  "Waehrung": "EURO",
  "Inhaber":
  {
    "Name": "Mustermann",
    "Vorname": "Max",
    "maennlich": true,
    "Hobbys": [ "Reiten", "Golfen", "Lesen" ],
    "Alter": 42,
    "Kinder": [],
    "Partner": null
  }
}
```

Konvertierung: Python Daten $\Leftrightarrow$ JSON

Der Inhalt von Python-Variablen kann als JSON in einer Text-Datei gespeichert werden.

```
import json
l = ['foo', {'bar': ('baz', None, 1.0, 2)}]
with open('testfile.json', 'w') as modelfile:
    json.dump(l, modelfile)
```

- Inhalt von `testfile.json`:
  `["foo", {"bar": ["baz", null, 1.0, 2]}]`
- JSON von Datei lesen:

```
with open('testfile.json', 'r') as modelfile:
    l_reloaded = json.load(modelfile)
```

- Ergebnis von:
  `l == l_reloaded`
  `l is l_reloaded`
  ?

Zusammenfassung

- Sentiment-Analyse
  - Regelbasiert vs. Maschinelles Lernen
  - Entitätenzentriert vs. Dokumenten-Ebene
- Perzeptron
  - Bei Fehlklassifikation einer Instanz ...
  - ... werden die Merkmalsgewichte entsprechend korrigiert
  - mehrere Iterationen über Trainingsset
  - Optimale Anzahl der Iterationen wird durch Entwicklungsdaten (dev-Set) bestimmt