Machine-learning Basics; Dateninstanzen in Python; Unit Tests

Benjamin Roth

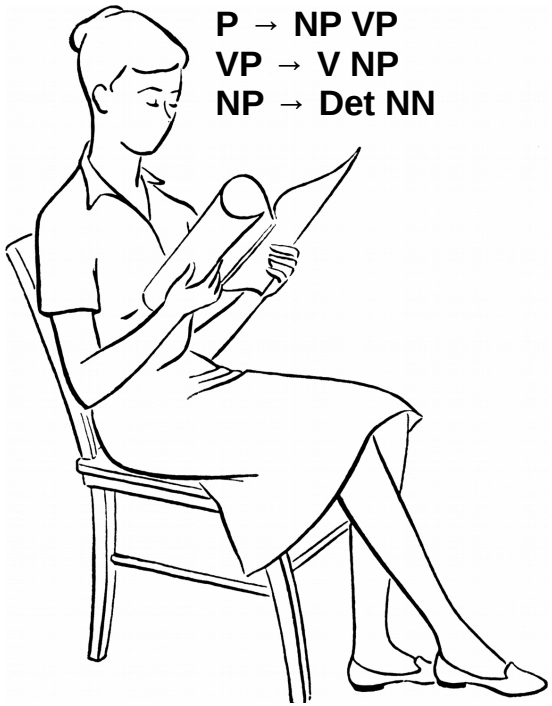
CIS LMU München

Übersicht

Übersicht

Maschinelles Lernen

- Maschinelle Lernverfahren für die Sprachverarbeitung
  - Wozu?
  - Vorteile und Nachteile gegenüber Alternativen?
  - Genauigkeit; Abdeckung; erforderliche Ressourcen (Daten, Expertise, Arbeitsaufwand).



Übersicht

Maschinelles Lernen - Eine Definition

"A computer program is said to learn from **experience** E with respect to some class of **tasks** T and **performance measure** P, if its performance at tasks in T, as measured by P, improves with experience E."
(Mitchell 1997)

Maschinelles Lernen - Eine Definition

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."
(Mitchell 1997)

- Lernen: Die Fähigkeit erlangen, eine Aufgabe auszuführen.
- Aufgaben werden als Menge von **Beispielen** repräsentiert (*"experience"*).
- Beispiele werden durch **Merkmale** (Features) repräsentiert: Mengen numerischer Eigenschaften, die als Vektoren $\boldsymbol{x} \in \mathbb{R}^n$ dargestellt werden können.

Übersicht

Daten

"A computer program is said to learn from **experience** E [...], if its performance [...] improves with **experience** E."

- Datensatz: Sammlung von Beispielen
- Design-Matrix

$$\boldsymbol{X} \in \mathbb{R}^{n \times m}$$

  - n: Anzahl von Beipielen
  - m: Anzahl von Merkmalen
  - Beispiel: $X_{i,j}$ Wert des Merkmals j (z.B. Häufigkeit eines Wortes) in Dokument i.
- Bei überwachten Lernverfahren, z.B. Klassifizierung, gibt es für jedes Beispiel noch ein **Label**
  - Label: vorauszusagenden Wert/Kategorie
  - Die Labels werden aus den Features $\boldsymbol{X}$ vorhergesagt.
  - Trainingsdaten: Labelvector $\boldsymbol{y} \in \mathbb{R}^n$ zusätzlich zu $\boldsymbol{X}$

Datensatz

Email Betreff	Label (good email?)
y 2 k - texas log	True
emerging small cap	False
re : patchs work better then pillz	False
meter 1431 - nov 1999	True
re : lyondell citgo	True
dobmeos with hgh my energy level has gone up	False
re : entex transistion	True
your prescription is ready . . oxwq s f e	False
get that new car 8434	False
entex transistion	True
unify close schedule	True
await your response	False

Merkmale

- Durch welche *Merkmale* könnte jede Instanz (Email) dargestellt werden?
- Wie können die Merkmale einer Instanz als *Dictionary* repräsentiert werden?
- Wie können die Merkmale einer Instanz als *Vektor* dargestellt werden?

Merkmale

- Durch welche *Merkmale* könnte jede Instanz (Email) dargestellt werden?
  - z.B. Unigramme, Bigramme, Shape-Features ....
- Wie können die Merkmale einer Instanz als *Dictionary* repräsentiert werden?
  - Abbildung Merkmal $\rightarrow$ Ausprägung (z.B. Häufigkeit) des Merkmals. (*feature* $\rightarrow$ *feature value*)
- Wie können die Merkmale einer Instanz als *Vektor* dargestellt werden?
  - Jede Vektor-Komponente entspricht einem möglichen Merkmal. Wenn ein Merkmal vorhanden ist, hat Vektor an dieser Stelle den Wert der Ausprägung, ansonsten den Wert 0.
- Eine Instanz besteht aus
  - Merkmalen mit ihren Ausprägungen
  - Label
- Ein Datensatz besteht aus einer Menge von Instanzen
- alternativ: Ein Datensatz besteht aus Design-Matrix und Label-Vektor

Übersicht

Aufgaben/Problemklassen

"A computer program is said to learn [...] with respect to some class of **tasks T** [...] if its performance at **tasks in T** [...] improves [...]"

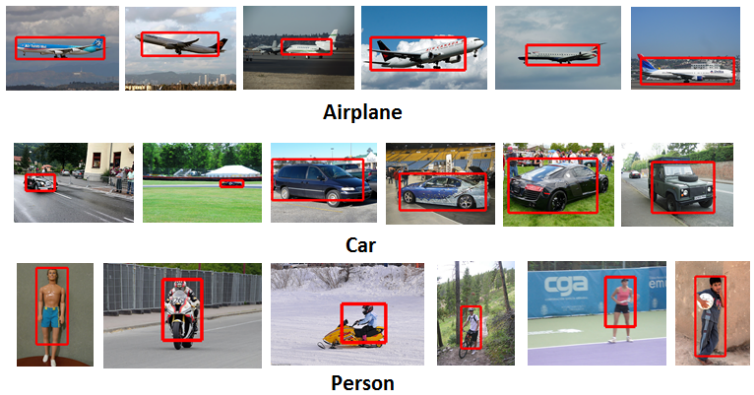
- Klassifizierung
- Regression
- Strukturvorhersage
- Erkennen von Anomalie
- Synthese und Sampling
- Vorhersage fehlender Werte
- Entstörung
- Clustering
- ...

Aufgabe: Klassifizierung

- Zu welcher von k Kategorien gehört ein Beispiel?

$$f : \mathbb{R}^n \to \{1 \dots k\}$$

- Beispiel: Kategorisierung von Bildausschnitten
  - Merkmalsvektor: Farbanteile für jedes Pixel; Davon abgeleitete Merkmale.
  - Vordefinierte Menge von Ausgabekategorien



- Beispiel: Erkennung von Spam Emails
  - Merkmalsvektor: Hochdimensionaler Vektor mit wenigen Einträgen $\neq 0$ (sparse).
    Jede Dimension zeigt das Vorkommen eines bestimmten Wortes an.
  - Ausgabekategorien: *"Spam Email"* vs. *'Kein Spam"*

Klassifizierung - wichtige Konzepte

- Lineares Modell
- Fehlerfunktion
- Overfitting
- Regularisierung
- Beispiele von Klassifikatoren

Übersicht

Performanz-Maße (Fehlerfunktionen)

"A computer program is said to learn [...] with respect to some [...] **performance measure** P, if its performance [...] **as measured by** P, improves [...]"

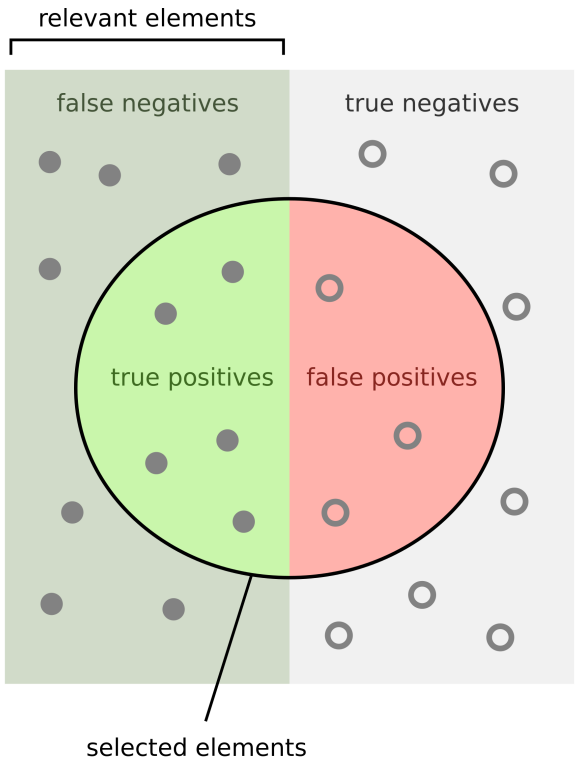
- Ein Performanz-Maß ermöglicht die Vorhersagequalität eines Algorithmus' quantitativ festzustellen.
- Welches Maß verwendet werden kann, hängt von der Art der Aufgabe ab:
  - Klassifikation: Accuracy, F1-Score
  - Ranking: Mean Average Precision, Spearman's Rank Correlation
  - Regression: Mean Squared Error
  - Textüberlappung (maschinelle Übersetzung): BLEU, ...
  - Bei Wahrscheinlichkeitsmodellen kann immer auch die Likelihood (Wahrscheinlichkeit der Daten) als Maß verwendet werden.

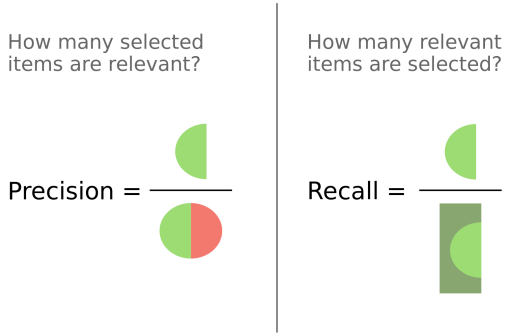
Fehlerfunktionen für Klassifikation

- Accuracy
  - Anteil der Instancen, für die der Klassifikator die korrekte Kategorie vorhersagt.
  - 0-1 loss = error rate = 1 - accuracy.
- Wenn ein großes Ungleichgewicht in der Verteilung der klassen besteht (relevante Kategorie ist selten), ist das F-measure besser geeignet:

$$\text{F1-score} = \frac{2 \cdot \text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}}$$

- Beispiel: in einem Datensatz sin 99% der Emails *GUT*, und 1% *SPAM*. Warum ist das F-measure hier ein besseres Maß als die Accuracy?





F-measure (=F-score)

- F-measure
  - wird berechnet aus Precision und Recall:

$$\text{F1-score} = \frac{2 \cdot \text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}}$$

$$\text{Prec} = \frac{|\text{relevant} \cap \text{retrieved}|}{|\text{retrieved}|}$$

$$\text{Rec} = \frac{|\text{relevant} \cap \text{retrieved}|}{|\text{relevant}|}$$

  - "relevant": Menge der Instanzen, **die** das relevante Label **haben**
  - "retrieved": Menge der Instanzen, **für die** das relevante Label **vorhergesagt wurde**.

Übersicht

Aufteilung der Daten

- Erster Ansatz: Daten in Trainings und Testdatensatz aufteilen.

Email Betreff	Label
y 2 k - texas log	1
emerging small cap	0
re : patchs work better then pillz	0
meter 1431 - nov 1999	1
re : lyondell citgo	1
dobmeos with hgh my energy level has gone up	0
re : entex transistion	1
your prescription is ready . . oxwq s f e	0
get that new car 8434	0
entex transistion	1
unify close schedule	1
await your response	0

Auswahl eines Modells

- Wahrscheinlich ist der Trainingsfehler $J_{train}(\theta)$ kleiner als der Testfehler $J_{test}(\theta)$.
- Folgende Modelle werden durchprobiert:
  - 100 Merkmale
  - 1000 Merkmale
  - 10000 Merkmale
  - ...
- Option 1: Optimiere Parameter für jedes der Modelle (anhand Trainingsset), und wähle Model anhand der Fehlerquote auf dem Test-set.
- Angenommen das Modell mit 1000 Merkmalen gibt das beste Ergebnis.
- Ist die Fehlerquote auf den Testdaten eine korrekte Schätzung der in Zukunft zu erwartenden Fehlerrate?

Auswahl eines Modells

- Wahrscheinlich ist der Trainingsfehler $J_{train}(\theta)$ kleiner als der Testfehler $J_{test}(\theta)$.
- Folgende Modelle werden durchprobiert:
  - 100 Merkmale
  - 1000 Merkmale
  - 10000 Merkmale
  - ...
- Option 1: Optimiere Parameter für jedes der Modelle (anhand Trainingsset), und wähle Model anhand der Fehlerquote auf dem Test-set.
- Angenommen das Modell mit 1000 Merkmalen gibt das beste Ergebnis.
- Ist die Fehlerquote auf den Testdaten eine korrekte Schätzung der in Zukunft zu erwartenden Fehlerrate?
- Antwort: Nein. Der zusätzliche Parameter "Anzahl der Merkmale" ist auf das Testset überangepasst.

Auswahl eines Modells

- Besser: Daten in Trainings-, Kreuzvalidierungs- and Testdaten aufteilen (z.B. 60%–20%–20%).
- Kreuzvalidierungsdaten werden auch Entwicklungsdaten genannt (cross-validation set, development set).

Subject	Label
y 2 k - texas log	1
emerging small cap	0
re : patchs work better then pillz	0
meter 1431 - nov 1999	1
re : lyondell citgo	1
dobmeos with hgh my energy level has gone up	0
re : entex transistion	1
your prescription is ready . . oxwq s f e	0
get that new car 8434	0
entex transistion	1
unify close schedule	1
await your response	0

Trainings- / Kreuzvalidierungs- / Test-Fehler

- Merkmalsgewichte werden auf Trainingsdaten geschätzt.
- Das Modell (Merkmale, Hyperparameter) wird anhand der Development-Daten ausgewählt.
- Die zu erwartende Performanz des Models wird anhand der Testdaten ermittelt.
- Ergebnisse auf Trainings- or Kreuzvalidierungsdaten können nicht als Berwertung des Algorithmus aufgefasst werden!

Übersicht

Merkmale für Dokumente

- Dokument: Tweet, Wikipedia-Artikel, Email, ...

id:doc1

---

text:
The raw text string of the document
The tokenized text list of strings
The token frequencies of the documents A unique identifier for each document

Merkmale für Dokumente

- Wie oft kommt jedes Wort vor?

id:doc1

---

text:
The raw text string of the document
The tokenized text list of strings
The token frequencies of the documents A unique identifier for each document

'the': 5
'of': 3
'text', 2
'document', 2
'for', 1
...

Merkmale für Dokumente

- Wie oft kommt jedes Bigram vor?

id:doc1

---

text:
The raw text string of the document
The tokenized text list of strings
The token frequencies of the documents A unique identifier for each document

'of the': 2
'the document': 2
'string of', 1
...

Merkmale für Dokumente

- Wie oft kommt jedes Zeichen-Ngram (z.B. Trigram) vor?

id:doc1

---

text:
The raw text string of the document
The tokenized text list of strings
The token frequencies of the documents A unique identifier for each document

'the': 5
'doc': 3
'ocu', 3
'ing', 2
...

Trainings-, Development- oder Test-Instanz

- Merkmale → Merkmalsausprägungen
- Label

```
class DataInstance:
    def __init__(self, feature_counts, label):
        # feature counts: dictionary (string -> int)
        self.feature_counts = feature_counts
        # label: True or False
        self.label = label

    @classmethod
    def from_list_of_feature_occurrences(cls, feature_list,
                                         label):
        feature_counts = dict()
        # TODO: count how often a feature occurs in the list.
        # ...
        return cls(feature_counts, label)
```

Trainings-, Dev- oder Test-Set

- Menge der möglichen Merkmalsausprägungen ist z.B. für Glättung wichtig.

```
class Dataset:
    def __init__(self, instance_list, feature_set):
        self.instance_list = instance_list
        self.feature_set = feature_set

    # ...
```

Trainings-, Dev- oder Test-Set

- Alternativ kann der Konstruktor die Merkmale auch aus den Instanzen übernehmen.

```
class Dataset:
    def __init__(self, instance_list):
        """ A data set is defined by a list of instances """
        self.instance_list = instance_list
        self.feature_set = \
            set.union(*[set(inst.feature_counts.keys()) \
                        for inst in instance_list])
```

Trainings-, Dev- oder Test-Set

- Welches sind die Merkmale, die in den meisten Instanzen vorkommen? Oft ist es sinnvoll, nur die häufigsten Merkmale (z.B. 1000) zu verwenden.
- Ein Datenset kann auf eine bestimmte Merkmalsmenge eingeschränkt werden. Andere Merkmale werden dann aus den Instanzen des Datensatzes entfernt

```
class Dataset:
    def __init__(self, instance_list, feature_set):
        self.instance_list = instance_list
        self.feature_set = feature_set
    def get_topn_features(self, n):
        # ...
    def set_feature_set(self, feature_set):
        # ...
```

Trainings-, Dev- oder Test-Set

- Sanity-check: Welche Genauigkeit hätte Vorhersage der häufigsten Kategorie?
- Manche Lernalgorithmen verlangen mehrere Trainings-Iterationen, zwischen denen das Trainingsset neu permutiert (gemischt) werden sollte.

```
class Dataset:
    def __init__(self, instance_list, feature_set):
        self.instance_list = instance_list
        self.feature_set = feature_set
    def most_frequent_sense_accuracy(self):
        # ...
    def shuffle(self):
        # ...
```

Übersicht

Unit Testing: Motivation

- It is unavoidable to have errors in code.
- Unit-testing helps you ...
  - ... to catch certain errors that are easy to automatically detect.
  - ... to be more clear about the specification of the intended functionality.
  - ... to be more stress-free when developing.
  - ... to check that functionality does not change when you re-organize or optimize code.
- Today, we will lok at two frameworks for unit testing that come prepackaged with Python
  1. `doctest`: A simple testing framework, where example function calls (together with their expected output) are written into the docstring documentation, and then are automatically checked.
  2. `unittest`: A framework, where several tests can be grouped together, and that allows for more complex test cases.

Test-Driven Development (TDD)

- Write tests first (, implement functionality later)
- Add to each test an empty implementation of the function (use the `pass`-statement)
- The tests initially all fail
- Then implement, one by one, the desired functionality
- Advantages:
  - Define in advance what the expected input and outputs are
  - Also think about important boundary cases (e.g. empty strings, empty sets, `float(inf)`, 0, unexpected inputs, negative numbers)
  - Gives you a measure of progress (*"65% of the functionality is implemented"*) - this can be very motivating and useful!

The `unittest` module

- Similar to Java's *JUnit* framework.
- Most obvious difference to `doctest`: test cases are not defined inside of the module which has to be tested, but in a separate module just for testing.
- In that module ...
  - `import unittest`
  - import the functionality you want to test
  - define a class that inherits from `unittest.TestCase`
    - ★ This class can be arbitrarily named, but `XyzTest` is standard, where `Xyz` is the name of the module to test.
    - ★ In `XyzTest`, write member functions that start with the prefix `test...`
    - ★ These member functions are automatically detected by the framework as tests.
    - ★ The tests functions contain `assert`-statements
    - ★ Use the `assert`-functions that are inherited from `unittest.TestCase` (do not use the Python built-in `assert` here)

Different types of asserts

Method	Checks that	New in
`assertEqual(a, b)`	`a == b`	
`assertNotEqual(a, b)`	`a != b`	
`assertTrue(x)`	`bool(x) is True`	
`assertFalse(x)`	`bool(x) is False`	
`assertIs(a, b)`	`a is b`	3.1
`assertIsNot(a, b)`	`a is not b`	3.1
`assertIsNone(x)`	`x is None`	3.1
`assertIsNotNone(x)`	`x is not None`	3.1
`assertIn(a, b)`	`a in b`	3.1
`assertNotIn(a, b)`	`a not in b`	3.1
`assertIsInstance(a, b)`	`isinstance(a, b)`	3.2
`assertNotIsInstance(a, b)`	`not isinstance(a, b)`	3.2

Question: ... what is the difference between "`a == b`" and "`a is b`"?

Example: using unittest

- test_square.py

```
import unittest
from example_module import square

class SquareTest(unittest.TestCase):
    def testCalculation(self):
        self.assertEqual(square(0), 0)
        self.assertEqual(square(-1), 1)
        self.assertEqual(square(2), 4)
```

Example: running the tests initially

- test_square.py

```
$ python3 -m unittest -v test_square.py
testCalculation (test_square.SquareTest) ... FAIL

======================================================================
FAIL: testCalculation (test_square.SquareTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/ben/tmp/test_square.py", line 6, in testCalculation
    self.assertEqual(square(0), 0)
AssertionError: None != 0

----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (failures=1)
$
```

Example: running the tests with implemented functionality

```
$ python3 -m unittest -v test_square.py
testCalculation (test_square.SquareTest) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
$
```

SetUp and Teardown (optional)

- `setUp` and `teardown` are recognized and exectuted automatically before (after) each unit test is run.
- `setUp`: Establish pre-conditions that hold for several tests.
  Examples:
  - Prepare inputs and outputs
  - Establish network connection
  - Read in data from file
- `tearDown` (less frequently used): Code that must be executed after tests finished
  Example: Close network connection

Example using setUp and tearDown

```
class SquareTest(unittest.TestCase):
    def setUp(self):
        self.inputs_outputs = [(0,0),(-1,1),(2,4)]

    def testCalculation(self):
        for i,o in self.inputs_outputs:
            self.assertEqual(square(i),o)

    def tearDown(self):
        # Just as an example.
        self.inputs_outputs = None
```

Summary

- Test-driven development
- Using `unittest` module
- Also have a look at the online documentation:
  `https://docs.python.org/3/library/unittest.html`
- Questions?