Machine learning for natural language processing

Reading course, Spring 2014

UNIVERSITY OF GOTHENBURG

Språk-
BANKEN

Richard Johansson

January 28, 2014

overview

practical matters

a motivating example

a quick overview of the suggested reading topics

this course

- machine learning for natural language processing
- you have different backgrounds: some are beginners and some are already experienced
  - this is an opportunity to complete your knowledge of ML or more NLP, or both
  - as long as you are able to write and present comprehensibly for a general audience in the end

possible organization of the course: up for discussion!

- introduction (today)
- reading period
- seminars
- writing papers

examination

- write a paper
  - something like a conference paper
  - (optionally) submit it to a conference
- present it at a seminar

course webpage

- `http://spraakbanken.gu.se/swe/personal/richard/mlnlp`
- contains practical information regarding the course

study material

- books
- papers
- Nivre's lectures

overview

practical matters

a motivating example

a quick overview of the suggested reading topics

in general: what is machine learning?

- assume we want to build some **prediction function**:
  - given a document, say whether it is sports-related or not
  - given an image, say whether or not it contains a human face
  - given a sentence, predict its syntactic analysis
- instead of hand-coding the function, some part of it is tuned automatically (**learned**) from **examples**
- central theoretical challenge: how to induce "knowledge" from given examples, so we can make predictions in new situations

example: grouping documents by customer sentiment

- our task: develop a program that groups customer reviews into positive and negative classes (given the text only)

★☆☆☆☆ **Just plain lame.**, August 14, 2007

By **Gary Smith "Editor, Handgun Hunter Magazine"** (Texas) - See all my reviews

This review is from: Garden & Gun (Magazine)

This magazine has a catchy title and very nice graphics and photography. What the premier issue lacks is anything of any substance about guns or hunting. I wonder if they actually read their own title. In my opinion these guys are nothing more than posers from the guns/hunting standpoint and many of the photographs appear to be staged. In particular, there are a couple pictures of a woman shooting a bow and arrow. Not only is she showing extremely poor form she's using the equipment shown in the photographs incorrectly. This is tantamount to using spinning gear with the reel positioned over the top of the fishing pole. If they want to cover hunting they should at least hire a photo editor that knows what (s)he's looking at. If you want a hunting magazine buy something else...

one approach: use a sentiment wordlist

- . . . for instance the MPQA list

```
type=strongsubj len=1 word1=wretchedly pos1=anypos stemmed1=n priorpolarity=negative
type=strongsubj len=1 word1=wretchedness pos1=noun stemmed1=n priorpolarity=negative
type=weaksubj len=1 word1=writhe pos1=verb stemmed1=y priorpolarity=negative
type=weaksubj len=1 word1=wrong pos1=adj stemmed1=n priorpolarity=negative
type=weaksubj len=1 word1=wrong pos1=anypos stemmed1=y priorpolarity=negative
type=weaksubj len=1 word1=wrongful pos1=adj stemmed1=n priorpolarity=negative
type=strongsubj len=1 word1=wrongly pos1=anypos stemmed1=y priorpolarity=negative
type=weaksubj len=1 word1=wrought pos1=adj stemmed1=n priorpolarity=negative
type=weaksubj len=1 word1=wrought pos1=noun stemmed1=n priorpolarity=negative
type=strongsubj len=1 word1=wry pos1=adj stemmed1=n priorpolarity=positive
type=strongsubj len=1 word1=yawn pos1=noun stemmed1=n priorpolarity=negative
type=strongsubj len=1 word1=yawn pos1=verb stemmed1=y priorpolarity=negative
type=strongsubj len=1 word1=yeah pos1=anypos stemmed1=y priorpolarity=neutral
type=strongsubj len=1 word1=yearn pos1=verb stemmed1=y priorpolarity=positive
type=strongsubj len=1 word1=yearning pos1=noun stemmed1=n priorpolarity=positive
type=strongsubj len=1 word1=yearningly pos1=anypos stemmed1=n priorpolarity=positive
type=strongsubj len=1 word1=yelp pos1=verb stemmed1=y priorpolarity=negative
type=strongsubj len=1 word1=yep pos1=anypos stemmed1=y priorpolarity=positive
type=strongsubj len=1 word1=yes pos1=anypos stemmed1=y priorpolarity=positive
type=weaksubj len=1 word1=youthful pos1=adj stemmed1=n priorpolarity=positive
type=strongsubj len=1 word1=zeal pos1=noun stemmed1=n priorpolarity=positive
type=strongsubj len=1 word1=zealot pos1=noun stemmed1=n priorpolarity=negative
type=strongsubj len=1 word1=zealous pos1=adj stemmed1=n priorpolarity=negative
type=strongsubj len=1 word1=zealously pos1=anypos stemmed1=n priorpolarity=negative
type=strongsubj len=1 word1=zenith pos1=noun stemmed1=n priorpolarity=positive
type=strongsubj len=1 word1=zest pos1=noun stemmed1=n priorpolarity=positive
```

document sentiment by summing word scores

- store all MPQA sentiment values in a table as numerical values
- e.g. 2 points for strong positive, -1 point for weak negative
- predict the overall sentiment value of the document by summing the scores of each word occurring

```
def guess_sentiment(document, weights):
    score = 0
    for word in document:
        score += weights.get(word, 0)
    if score >= 0:
        return "pos"
    else:
        return "neg"
```

experiment

- we evaluate on 50% of a sentiment dataset
  `http://www.cs.jhu.edu/~mdredze/datasets/sentiment/`

```
def evaluate(labeled_documents, weights):
    ncorrect = 0
    for _, label, _, document in labeled_documents:
        guess = guess_sentiment(document, weights)
        if guess == label:
            ncorrect += 1
    return float(ncorrect) / len(labeled_documents)
```

- this is a balanced dataset, coin-toss accuracy would be 50%
- with MPQA, we get an accuracy of 59.5%

can we do better?

- it's hard to set the word weights
- what if we don't even have a resource such as MPQA?
- can we set the weights automatically?

an idea for setting the weights automatically

- start with an empty weight table (instead of using MPQA)
- classify documents according to the current weight table
- each time we misclassify, change the weight table a bit
  - if a positive document was misclassified, add 1 to the weight of each word in the document
  - and conversely . . .

```
def train_by_errors(labeled_documents, number_iterations):
    weights = {}
    for iteration in range(number_iterations):
        for _, label, _, document in labeled_documents:
            guess = guess_sentiment(document, weights)
            if label == "pos" and guess == "neg":
                for word in document:
                    weights[word] = weights.get(word, 0) + 1
            elif label == "neg" and guess == "pos":
                for word in document:
                    weights[word] = weights.get(word, 0) - 1
    return weights
```

new experiment

- we compute the weights using 50% of the sentiment data and test on the other half
- the accuracy is 81.4%, up from the 59.5% we had when we used the MPQA
- machine learning doesn't have to be harder than this!
  - `train_by_errors` is called the **perceptron** algorithm and is one of the most widely used machine learning algorithms

examples of the weights

amazing 171	waste -175
easy 124	worst -168
perfect 109	boring -154
highly 108	poor -134
five 107	‘ -130
excellent 104	unfortunately -122
enjoy 93	horrible -118
job 92	ok -111
question 90	disappointment -109
wonderful 90	unless -108
performance 83	called -103
those 80	example -100
r&b 80	bad -100
loves 79	save -99
best 78	bunch -98
recommended 77	talk -96
favorite 77	useless -95
included 76	author -94
medical 75	effort -94
america 74	oh -94

a geometric view of the same thing

- in the literature, typically a geometric formulation is used: given some object x, compute the score

$$w \cdot f(x)$$

  and return "positive" if the score > 0, otherwise "negative"
- $f(x)$ is a **feature vector** representing the object x, and w is a **weight vector** returned by the learning procedure
  - for instance, the document "useless waste" might be

$$[0, 0, 0, 1, 0, 0, 1, 0, 0, 0, \ldots]$$

  - ... and w_4 could be -96 and w_7 -175
- in document classification, the dimensions typically correspond to word types
- in general, x can be any object, not just a document

the same thing with scikit-learn

- to train a classifier:

```
vec = DictVectorizer()
clf = Perceptron(n_iter=20)
clf.fit(vec.fit_transform(train_docs),
        numpy.array(train_targets))
```

- to classify a new instance:

```
guess = clf.predict(vec.transform(doc))
```

other types of classifiers

- there are many other approaches apart from the perceptron
- `http://scikit-learn.org/stable/supervised_learning.html`
- why?

what if we have more than two categories?

- for instance `['dvd', 'books', 'health', 'music', 'camera', 'software']`
- a possible solution: instead of one weight for each word, one weight for each word–category pair

```
def score_category(document, weights, category):
    score = 0
    for word in document:
        score += weights.get((word, category), 0)
    return score

def guess_category(document, weights, categories):
    return max(categories, key=lambda c: score_category(document, weights, c))
```

perceptron training with multiple categories

- if a document is misclassified: increase the weights for the correct label and decrease them for the incorrect guess

```
def train_by_errors(labeled_documents, categories, number_iterations):
    weights = {}
    for iteration in range(number_iterations):
        for label, _, _, document in labeled_documents:
            guess = guess_category(document, weights, categories)
            if label != guess:
                for word in document:
                    weights[(word, label)] = weights.get((word,label), 0) + 1
                    weights[(word, guess)] = weights.get((word,guess), 0) - 1
    return weights
```

a geometric view (multiple categories)

- given some object x, for each possible output y compute the score

$$w \cdot f(x, y)$$

  and return the highest-scoring y
- $f(x, y)$ is a feature representation of the input–output pair

structured prediction

- given an input x (e.g. a sentence), predict an output y (e.g. a tag sequence or a parse tree)
- a typical solution is similar to what we did in the multiple-category case: compute a score defined as

$$w \cdot f(x, y)$$

  and maximize w.r.t. y
- . . . but the maximization step depends on the task, e.g. Viterbi for taggers and CKY for parsers

overview

practical matters

a motivating example

a quick overview of the suggested reading topics