TEXT CLASSIFICATION WITH THE PERCEPTRON

Natural Language Processing
(COM4513/6513)

Nikos Aletras
n.aletras@sheffield.ac.uk
Department of Computer Science
University of Sheffield

TEXT CLASSIFICATION

A very common problem in NLP:

Given a piece of text, assign a label from a predefined set

What could the labels be?

- positive vs negative (e.g. sentiment in reviews)
- about world politics or not
- author name (author identification)
- pass or fail in essay grading

IN THIS LECTURE

We will see how to:

- representing documents as vectors
- learn a classifier using the perceptron

Ready for Lab in Week 2!

SENTIMENT ANALYSIS ON FILM REVIEWS

REPRESENTING TEXT

Raw text:

In [8]:
```
print(example_text[:500])
```

```
what's shocking about " carlito's way " is how good it is .
having gotten a bit of a bad rap for not being a big box office hit like pacin
o's previous film , " scent of a woman , " and not having as strong a performa
nce as he did in that one ( he had just won an oscar ) , " carlito's way " was
destined for underrated heaven .
that's what it is : an underrated gem of a movie .
and what a shame because pacino and de palma both do amazing jobs with it , an
d turn it into a great piece of a pulpy
```

Tokenised text:

In [11]:
```
print(re.sub("[^\w']"," ",example_text).split()[:100])
```

```
["what's", 'shocking', 'about', "carlito's", 'way', 'is', 'how', 'good', 'it',
'is', 'having', 'gotten', 'a', 'bit', 'of', 'a', 'bad', 'rap', 'for', 'not',
'being', 'a', 'big', 'box', 'office', 'hit', 'like', "pacino's", 'previous',
'film', 'scent', 'of', 'a', 'woman', 'and', 'not', 'having', 'as', 'strong',
'a', 'performance', 'as', 'he', 'did', 'in', 'that', 'one', 'he', 'had', 'jus
t', 'won', 'an', 'oscar', "carlito's", 'way', 'was', 'destined', 'for', 'under
rated', 'heaven', "that's", 'what', 'it', 'is', 'an', 'underrated', 'gem', 'o
f', 'a', 'movie', 'and', 'what', 'a', 'shame', 'because', 'pacino', 'and', 'd
e', 'palma', 'both', 'do', 'amazing', 'jobs', 'with', 'it', 'and', 'turn', 'i
t', 'into', 'a', 'great', 'piece', 'of', 'a', 'pulpy', 'character', 'study',
"carlito's", 'way', 'deals']
```

Ideas?

Let's represent text with vectors. Why?

That's what machine learning algorithms take as input

COUNTING WORDS

In [4]:
```
dictionary = Counter(re.sub("[^\w']"," ",example_text).split()[:100])
print(dictionary)
```

```
Counter({'a': 9, 'it': 4, 'of': 4, 'and': 4, "carlito's": 3, 'way': 3, 'is':
3, 'having': 2, 'for': 2, 'not': 2, 'as': 2, 'he': 2, 'an': 2, 'underrated':
2, 'what': 2, "what's": 1, 'shocking': 1, 'about': 1, 'how': 1, 'good': 1, 'go
tten': 1, 'bit': 1, 'bad': 1, 'rap': 1, 'being': 1, 'big': 1, 'box': 1, 'offic
e': 1, 'hit': 1, 'like': 1, "pacino's": 1, 'previous': 1, 'film': 1, 'scent':
1, 'woman': 1, 'strong': 1, 'performance': 1, 'did': 1, 'in': 1, 'that': 1, 'o
ne': 1, 'had': 1, 'just': 1, 'won': 1, 'oscar': 1, 'was': 1, 'destined': 1, 'h
eaven': 1, "that's": 1, 'gem': 1, 'movie': 1, 'shame': 1, 'because': 1, 'pacin
o': 1, 'de': 1, 'palma': 1, 'both': 1, 'do': 1, 'amazing': 1, 'jobs': 1, 'wit
h': 1, 'turn': 1, 'into': 1, 'great': 1, 'piece': 1, 'pulpy': 1, 'character':
1, 'study': 1, 'deals': 1})
```

BAG OF WORDS REPRESENTATION

- The higher the counts for a word, the more important it is
- No document has every word; most have 0 counts (implicitly)
- For a given vobaculary, every document is represented by a vector of the same length

Anything missing?

- which words to keep?
- how to value their presence/absence?
- word order is ignored, could we add bigrams?

Choice of representation (features) matters a lot!

OUR FIRST CLASSIFIER

We represent a document as counts over words/features, $\mathbf{x} \in \Re^N$.

How to predict if it has positive $(y = 1)$ or negative $(y = -1)$ sentiment?

If each word n has counts x_n in the review and is associated with a weight (w_n), then:

$$\hat{y} = sign(\sum_{n=1}^{N} w_n x_n) = sign(\mathbf{w} \cdot \mathbf{x})$$

$$sign(\hat{y}) := \begin{cases} -1 & \text{if } \hat{y} < 0 \\ 1 & \text{otherwise} \end{cases}$$

$$\hat{y} = sign(\sum_{n=1}^{N} w_n x_n) = sign(\mathbf{w} \cdot \mathbf{x})$$

In [5]:
```
print(bag_of_words)
```

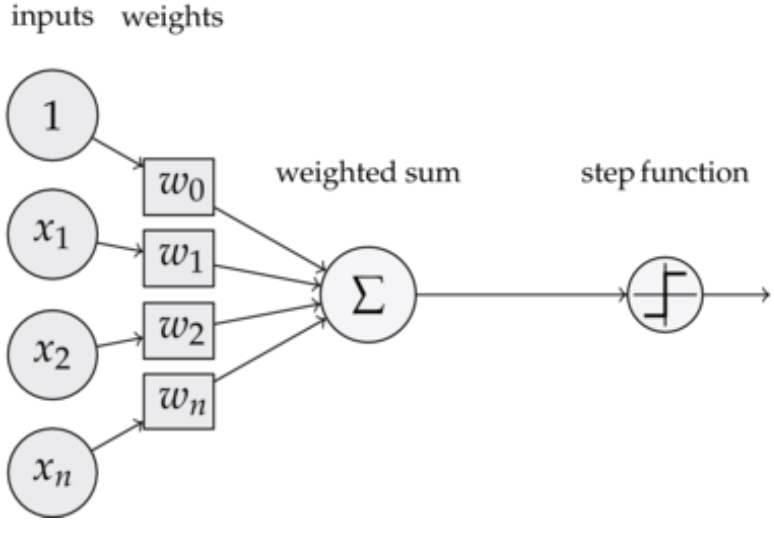
```
Counter({'and': 37, 'is': 26, 'he': 11, 'great': 10, 'carlito': 9, 'film': 8,
'but': 8, 'some': 7, 'pacino': 7, "carlito's": 7, 'palma': 5, 'well': 5, 'lik
e': 5, 'woman': 4, 'amazing': 4, 'bias': 1})
```

In [6]:
```
weights = dict({'and': 0.0, 'is': 0.0, 'he': 0.0, 'great': 0.0,\
                'carlito': 0.0, 'but': 0.0, 'film': 0.0, 'some': 0.0,\
                'carlito\'s': 0.0, 'pacino': 0.0, 'like': 0.0,\
                'palma': 0.0, 'well': 0.0, 'amazing': 0.0, 'woman': 0.0, 'bias':
 0.0})
```

In [7]:
```
score = 0.0
for word, counts in bag_of_words.items():
    score += counts * weights[word]
print(score)
print("positive") if score >= 0.0 else print("negative")
```

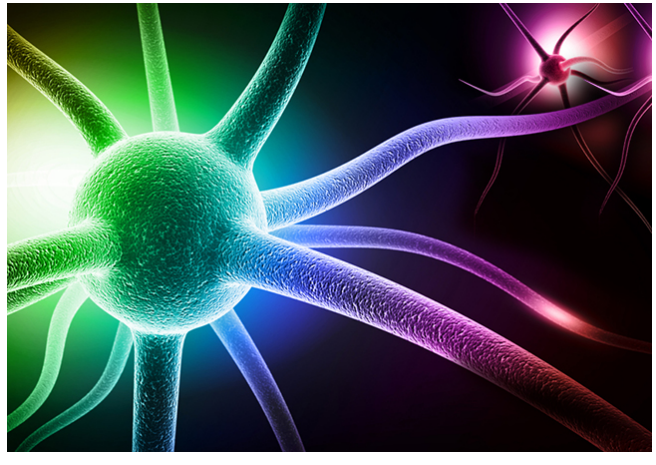
```
0.0
positive
```

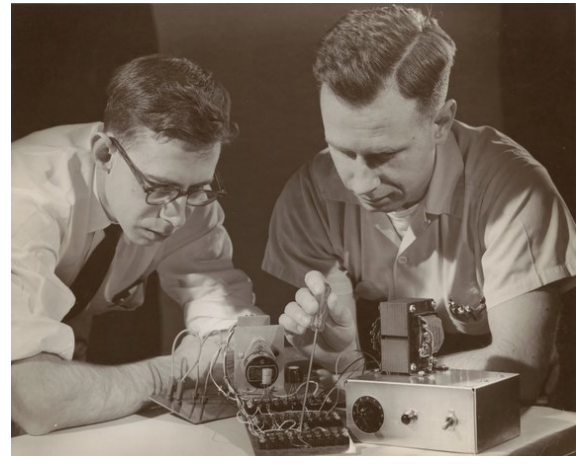
ANOTHER VIEW



How to learn the weights $\mathbf{w}$?

THE PERCEPTRON





Proposed by Rosenblatt in 1958 and still in use by researchers

SUPERVISED LEARNING

Given training documents with the correct labels

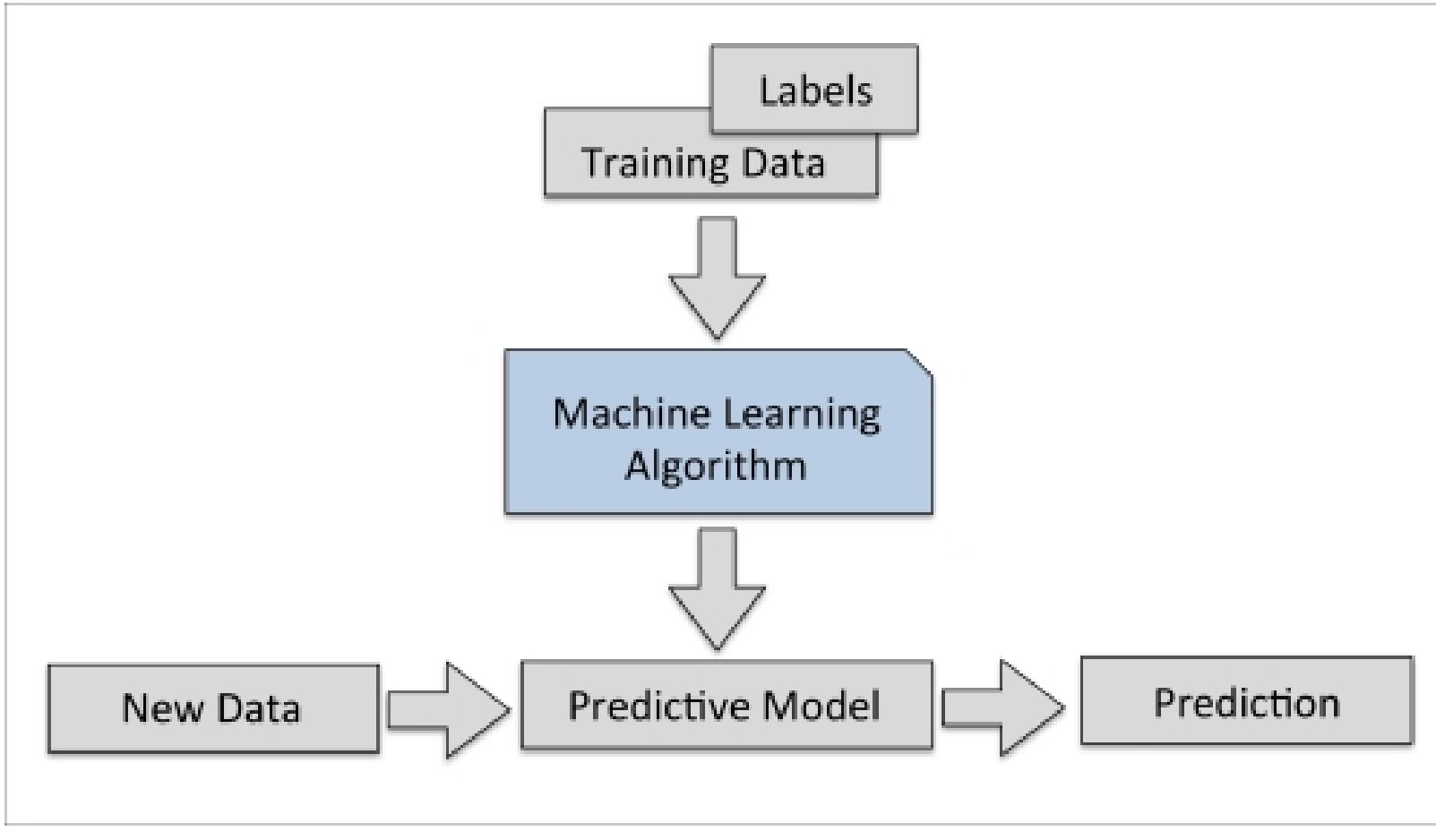
$$D_{train} = \{\mathbf{x}^1, y^1) \ldots (\mathbf{x}^M, y^M)\}$$

Find the weights $\mathbf{w}$ for the linear classifier

$$\hat{y} = sign(\sum_{n=1}^{N} w_n x_n) = sign(\mathbf{w} \cdot \mathbf{x})$$

so that we can predict the labels of **unseen** documents

SUPERVISED LEARNING



LEARNING WITH THE PERCEPTRON

Input: $D_{train} = \{(\mathbf{x}^1, y^1) \ldots (\mathbf{x}^M, y^M)\}$
set $\mathbf{w} = \mathbf{0}$
for $(\mathbf{x}, y) \in D_{train}$ **do**
    predict $\hat{y} = sign(\mathbf{w} \cdot \phi(\mathbf{x}))$
    if $\hat{y} \neq y$ **then**
        if $\hat{y}$ **is** 1 **then**
            update $\mathbf{w} = \mathbf{w} + \phi(\mathbf{x})$
        else
            update $\mathbf{w} = \mathbf{w} - \phi(\mathbf{x})$
return $\mathbf{w}$

- error-driven, online learning
- x is the document $\phi(x)$ is the bag of words, bigrams, etc.

A LITTLE TEST

Given the following tweets labeled with sentiment:

Label	Tweet
negative	Very sad about Iran.
negative	No Sat off...Need to work 6 days a week.
negative	I'm a sad panda today.
positive	such a beautiful satisfying day of bargain shopping. loves it.
positive	who else is in a happy mood??
positive	actually quite happy today.

What features would the perceptron find indicative of positive/negative class?

Would they generalize to unseen test data?

SPARSITY AND THE BIAS

In NLP, no matter how large our training dataset, we will never see (enough of) all the words/features.

- features unseen in training are ignored in testing
- there are ways to ameliorate this issue (e.g. word clusters), but it never goes away
- there will be texts containing only unseen words

Bias: that appears in each instance

- its value is hardcoded to 1
- that 1 in the diagram
- effectively learns to predict the majority class

3 TRICKS FOR BETTER PERCEPTRONS

averaging, multiple passes, shuffling

Input: $D_{train} = \{(x^1, y^1) \ldots (x^M, y^M)\}$

set $\mathbf{w}_0 = \mathbf{0}$; $c = 1$

for $i = 1$ **to** *maxIter* **do**

  $shuffle(D_{train})$

  for $(x, y) \in D_{train}$ **do**

    predict $\hat{y} = sign(\mathbf{w}_{c-1} \cdot \phi(x))$

    if $\hat{y} \neq y$ **then**

      update $\mathbf{w}_c = \mathbf{w}_{c-1} + y\phi(x)$

    else

      $\mathbf{w}_c = \mathbf{w}_{c-1}$

    $c = c + 1$

return $\frac{1}{c} \sum_{i=1}^{c} \mathbf{w}_i$

BINARY TO MULTICLASS

A vector of weights per label $y \in \mathcal{Y}$:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} (\mathbf{w}^y \cdot \phi(x))$$

Update rule:

$$\begin{aligned} &\textbf{if } \hat{y} \neq y \textbf{ then} \\ &\quad update\ \mathbf{w}^{\mathbf{y}} = \mathbf{w}^{\mathbf{y}} + \phi(\mathbf{x}) \\ &\quad update\ \mathbf{w}^{\hat{\mathbf{y}}} = \mathbf{w}^{\hat{\mathbf{y}}} - \phi(\mathbf{x}) \end{aligned}$$

Equivalently, make label-specific representations:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} (\mathbf{w} \cdot \phi(x, y))$$

OTHER POPULAR SUPERVISED ML ALGORITHMS

- Logistic Regression
- Support Vector Machines
- Naive Bayes
- Neural Networks (Week 8)
- Gaussian Processes

EVALUATION

The standard way to evaluate our classifier is:

$$Accuracy = \frac{\#\text{correct\_Labels}}{\#\text{all\_Instances}}$$

What could go wrong?

When one class is much more common than the other, predicting it always gives high accuracy.

EVALUATION

Predicted/Correct	MinorityClass	MajorityClass
MinorityClass	TruePositive	FalsePositive
MajorityClass	FalseNegative	TrueNegative

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

BIBLIOGRAPHY

- Manning, Raghavan and Schutze's vector space chapter from the Introduction to Information Retrieval.
- Hal Daumé III's chapter on the perceptron from his book on machine learning
- For more background reading on classification, Kevin Murphy's introduction touches upon most important concepts in ML

COMING UP NEXT WEEK

- So far we saw how to do text classification using

```
- a bag of words representation
- and the perceptron to learn a linear classifier
```

But we have ignored word order. Language is sequential! How we can develop sequential language models?