Classification with the Perceptron

L101: Machine Learning for Language Processing
Andreas Vlachos
L101 Practicalities

Lecturers:
- Ted Briscoe (ejb@cam.ac.uk)
- Andreas Vlachos (av308@cam.ac.uk)

Materials and information about the assessment:
https://www.cl.cam.ac.uk/teaching/2021/L101/
Why Natural Language Processing (NLP)?

Dan Jurafsky
Professor and Chair, Linguistics
Professor, Computer Science
Stanford University
Classification

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

What could the labels be?

- Positive/negative sentiment
- Topical
- Author name (authorship identification)
- Biased or not
- etc.
Formulation

Given input instance $x$, parameters $w$, a classifier is a function $f$ that predicts $\hat{y}$:

Binary (1/-1, sometimes 0/1): 
$$\hat{y} = \text{sign} f(x; w)$$

Multiclass (1 of N):
$$\hat{y} = \arg \max_{y \in \mathcal{Y}} f(x; w)$$

Multilabel (a set of labels, possibly empty):
$$\hat{y} = \arg \max_{y \in \mathcal{P}(\mathcal{Y})} f(x; w)$$
Features

Given a document, what features would you use to predict its:

- topic?
- sentiment?
- author name?
- factual correctness?

Some ideas:

- bag of words (n-grams)
- meta-data
- sub-word features
- external evidence, common sense...
- Often these are denoted with the feature function: $\phi(x) \in \mathcal{R}^k$
Binary linear classifier

\[ \hat{y} = \text{sign}(w \cdot \phi(x)) \]

The “linear” part refers to the function \( f \), a linear map:

\[ w \cdot \phi(x) \]

How do we learn the weights \( w \)?
Supervised learning

Given labeled training data of the form:

\[ D = \{ (x^1, y^1), \ldots, (x^M, y^M) \} \]

Learn weights \( w \) that generalize well to new instances
The perceptron

Proposed by Rosenblatt in 1958, still close to state-of-the-art
The perceptron

AI hype is not a new problem

We should always remember when we talk to the public
The perceptron algorithm

**Input:** training examples $\mathcal{D} = \{(x^1, y^1), \ldots (x^M, y^M)\}$

Initialize weights $w = (0, ..., 0)$

for $(x, y) \in \mathcal{D}$ do

Predict label $\hat{y} = \text{sign}(w \cdot \phi(x))$

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

Update $w = w + y\phi(x)$

end if

end for

error-driven, online learning
Testing our intuitions

Given the following tweets labeled with sentiment and assuming bag of words:

<table>
<thead>
<tr>
<th>Sentiment</th>
<th>Tweet</th>
</tr>
</thead>
<tbody>
<tr>
<td>negative</td>
<td>Very sad about Iran.</td>
</tr>
<tr>
<td>negative</td>
<td>No Sat off...Need to work 6 days a week.</td>
</tr>
<tr>
<td>negative</td>
<td>I’m a sad panda today.</td>
</tr>
<tr>
<td>positive</td>
<td>such a beautiful satisfying day of bargain shopping. loves it.</td>
</tr>
<tr>
<td>positive</td>
<td>who else is in a happy mood??</td>
</tr>
<tr>
<td>positive</td>
<td>actually quite happy today.</td>
</tr>
</tbody>
</table>

- What weights do you expect your perceptron to learn?
- Do you think they would generalize well?
Sparsity and 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, word vectors), but it never goes away
- there will be texts containing only unseen words

Bias: a feature that appears in each instance
- its value is hardcoded to 1
- often omitted from equations due to its omnipresence
- effectively learns to predict the majority class
Improving the perceptron

Input: training examples \( \mathcal{D} = \{(x^1, y^1), \ldots (x^M, y^M)\} \)

Initialize counter \( c = 0 \), weights \( w_c = (0, ..., 0) \)

for \( i = 1 \ldots maxIter \) do

Shuffle \( \mathcal{D} \)

for \( (x, y) \in \mathcal{D} \) do

Predict label \( \hat{y} = \text{sign}(w_c \cdot \phi(x)) \)

if \( \hat{y} \neq y \) then

Update \( w_{c+1} = w_c + y\phi(x) \)

else

\( w_{c+1} = w_c \)

end if

\( c = c + 1 \)

end for

end for

\( w = \frac{1}{c} \sum w_0 + \ldots + w_c \)

- Multiple passes
- Shuffling
- Averaging
On separating hyperplanes

If data not linearly separable, perceptron does not converge (will always update)
Even if linearly separable, no guarantee of finding a good separating hyperplane

Binary to multiclass

Binary: \[ \hat{y} = \text{sign}(w \cdot \phi(x)) \]

Multiclass 1: \[ \hat{y} = \arg \max_{y \in \mathcal{Y}} w_y \cdot \phi(x) \]
Input feature map: only describes \( x \).

Multiclass 2: \[ \hat{y} = \arg \max_{y \in \mathcal{Y}} w \cdot \phi(x, y) \]
Joint feature map: compatibility between \( x \) and \( y \). More expressive but less intuitive.
Multiclass perceptron (two versions)

Input: training examples $\mathcal{D} = \{(x^1, y^1), \ldots (x^M, y^M)\}$

Initialize weights $w_y = (0, \ldots, 0)$

Initialize weights $w = (0, \ldots, 0)$

for $(x, y) \in \mathcal{D}$ do

Predict label $\hat{y} = \arg \max_{y \in \mathcal{Y}} w_y \cdot \phi(x) / w \cdot \phi(x, y)$

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

Update $w_y = w_y + \phi(x)$

Update $w_{\hat{y}} = w_{\hat{y}} - \phi(x)$

Update $w = w + \phi(x, y) - \phi(x, \hat{y})$

end if

end for
Evaluation

\[ \text{accuracy} = \frac{\text{correctPredictions}}{\text{allPredictions}} \]

What can go wrong?

Imbalanced datasets: predicting one class gives 90% accuracy

Very common: most topics are not relevant to most documents, etc.
Evaluation

<table>
<thead>
<tr>
<th>Predicted/Correct</th>
<th>MinorityClass</th>
<th>MajorityClass</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinorityClass</td>
<td>TruePositive</td>
<td>FalsePositive</td>
</tr>
<tr>
<td>MajorityClass</td>
<td>FalseNegative</td>
<td>TrueNegative</td>
</tr>
</tbody>
</table>

\[
\text{Precision} = \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalsePositives}}
\]

\[
\text{Recall} = \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalseNegatives}}
\]

Often combined into their harmonic mean, aka F1-score
So far we assumed a fixed decision threshold but often it makes sense to adjust it.

Don’t pick a single threshold, check them all!

Can summarize in the area under the precision-recall curve (AUC)
Multilabel classification

\[ \hat{y} = \underset{y \in \mathcal{P}(Y)}{\text{arg max}} f(x; \omega) \]

Can be done in two ways:

- Binary relevance: build a binary classifier for each label in \( Y \)
  - Ignores label dependencies
- Multiclass: build a multiclass classifier for all members of the powerset \( \mathcal{P}(Y) \)
  - Can be computationally expensive, training data can be sparse

Both are instances of \textit{reduction}: transform complex problems to simpler ones

More \textit{advanced methods} exist, but always try these two first
What sentiment classifier would you build?

Type: Binary, multiclass, something else?

What features? What weights do you expect for them?
Bibliography

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