Sentiment Analysis; Classification with the Perceptron Algorithm

Benjamin Roth

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

Sentiment analysis

Sentiment analysis

- Sentiment analysis, Opinion Mining
- Automatic recognition of opinions, value judgments, evaluations, positive/negative emotions
- For an entire text or specifically in relation to a specific entity (product, subject, person, event ...)
- application e.g.
  - in business
  - in social research

Sentiment analysis: example

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**: Opinion about camera ABC
- **aspects**: Image quality, battery life, weight of the ABC Camera
- **opinion holder**: XYZ, wife of XYZ
- **sentiment**: Evaluation of the respective aspects by opinion holder
- **date**: September 10, 2011

Simplified problem definition

- Sentiment classification at document level.
- Problem: given an opinion document, determine the general sentiment of the opinion holder towards the entity.
- assumptions:
  - The document expresses exactly one sentiment.
  - The opinion holder ...
  - The entity ...
  - The time ...
  - The aspects ...
  - ... are known or irrelevant.
- e.g. given a movie rating (for example from IMDB) this rating is **positive** or **negative**
  - How could a 10-point rating be added to these categories?
  - How could a neutral category be predicted?

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.

Suggested solution: machine learning

- Do not use a given word list, but learn characteristics from annotated data.
- advantages:
  - Does not require man-made rules.
  - Find features that a human being would not think of.
  - Fine weighting of features.
  - Many different representations and classifiers possible (for example, Bag-of-words + Naive Bayes, Neural Networks, ...)
  - Can be linked to rule-based approach (for example, by feature: number of rules that match)
  - ...
- disadvantages:
  - Requires many annotated data instances.
  - Overfitting, domain dependency.
  - ...

Perceptron-Algorithm

Perceptron-Algorithm

- The perceptron algorithm falls into the group of **discriminative** models
- It optimizes the quality of the prediction.
- *"Given the features, what is the correct class?"*

Perceptron Algorithm: idea

- For each possible feature (e.g., word), a weight is learned (a real value that can be positive or negative).
- Prediction of the class for an instance:
  - For each feature, the value of the feature (e.g., occurrences in the document) is multiplied by its feature weight, and the results are summed.
  - If the resulting value is greater than 0, the positive class is predicted (POS), otherwise the negative class (NEG)
- The perceptron algorithm iteratively adjusts the feature weights to minimize misclassifications.

Representation of the instances as vectors

- Each instance i can be represented as a vector $x^{(i)}$ (as with the TF-IDF calculation).
- The number of components of the vector corresponds to the size of the feature space (e.g., the vocabulary).
- $x^{(i)} \in \mathbb{R}^n$ for a vocabulary of size n
- Each component of the vector corresponds to a word.

Example instance...

...as a vector for the vocabulary `[movie, entertain, highly, waste, time]`:

movie
time
waste

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

- Implementation: Since most entries are 0, we choose an efficient representation with dictionaries (word $\Rightarrow$ number)

Vector multiplication

- Repetition vector multiplication (whiteboard)

Vector multiplication with weight vector

- weight vector

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

- Predictive value (score) for instance i:

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

- Implementation: For the weight vector we also choose a representation with dictionaries (word $\Rightarrow$ weight)

Prediction

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

movie
time
waste

highly
entertain
movie

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

- Which weights would lead to a misclassification of both instances (any example)?
- How should these weights be corrected so that the instances are properly classified?

Prediction

- Decision rule:
  - $x^{(i)T} w > 0 \Rightarrow$ POS
  - $x^{(i)T} w \leq 0 \Rightarrow$ NEG
- Example:

movie
time
waste

highly
entertain
movie

  - $x^{(1)T} w = w_{\text{movie}} + w_{\text{waste}} + w_{\text{time}}$
  - $x^{(2)T} w = w_{\text{movie}} + w_{\text{entertain}} + w_{\text{highly}}$
- Weights that lead to a misclassification of both instances:
  z.B. $w_{\text{movie}} = 1.0; w_{\text{waste}} = -2.0; w_{\text{time}} = 1.5; w_{\text{entertain}} = -2.0; w_{\text{highly}} = 0.5$
- Correction of weights:
  - w_{waste} *and/or* w_{time} *must be decreased*
  - $w_{entertain}$ *and/or* w_{highly} *must be increased*
  - w_{movie}*: Draw for both instances.*
  - *all other weights neutral.*

Adjustment of weights (perceptron update)

- Idea:
  - if the prediction is already correct: do not do anything.
  - otherwise: Increase/decrease weights for each instance so that the score changes in the right direction.
    Attention: In our example, only positive feature values occurred, but the algorithm will also work with negative ones.
  - weights for features that occur particularly frequently with one of the two classes receive many adjustments in the respective direction.
  - the weights for features that appear roughly the same in both classes (for example, *"for"*) are sometimes changed in one direction and then in the other.

Perceptron update for one instance

- If the label matches, no weight update.
- else:
  - If predicted label is `True` and correct label is `False`: $error = 1$
    ($\Leftrightarrow$ reduce the weights for the occurring features)
  - if predicted label is `False` and correct label is `True`: $error = -1$
    ($\Leftrightarrow$ increase the weights for the occurring features)
  - Update for weight w_j (and feature value $x_j^{(i)}$)
    $w_j \leftarrow w_j - error \cdot x_j^{(i)}$

Perzceptron-Algorithm

- **Input:**
  - feature vectors (instances) $\{x^{(1)}, x^{(2)}, ...x^{(n)}\}$
  - labels $\{y^{(1)}, y^{(2)}, ...y^{(n)}\}$
- **Output:** feature weight vector w

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

Remarks

- Possible termination criteria:
  - Predefined number of iterations reached
  - Perfect classification on the training data
  - No error reduction on the development data
    $\Rightarrow$ take the weights from the last iteration.
- In our case, the feature values were integers (word occurrences), but the perceptron algorithm also works with non-integer and negative values.
- The perceptron algorithm converges to the perfect classification of the training data if they are linearly separable.
  - *linear separable*: There is a weight combination with perfect classification
  - not all training data are separable,
  - perfect classification on training data often generalizes badly

Hyper parameters for the perceptron algorithm

- Hyper parameters: Parameters that are given to the algorithm (= which it does not automatically learn) ...
- ... are selected on the development data
- What are hyperparameters for the perceptron algorithm?

Hyper parameters for the perceptron algorithm

- What features, number of features (for example, 1000 or 10000 most common words)
- Number of training iterations
  - Compare the accuracy on the development data after each iteration
  - Choose weight values of the iteration with the best development accuracy
  - "*Early-Stopping with patience n*": Cancel training if after $n+1$ iterations there is no improvement on the development data
- increment/size of updates no relevant hyper parameter (if decision threshold $= 0$)

Implementation

Perceptron Classifier: Constructors

- The only parameters of the perceptron classifier are the feature weights. ⇒ Dictionary.
- Classifier for a `dataset`: each property is initialized with 0.
  ⇒ The model still needs training.

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

Perceptron Classifier: Overview

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

prediction and update

- $\Rightarrow$ exercise
- decision rule for prediction?
- Update for an instance?

`prediction(counts)` and `update(instance)`

- $\Rightarrow$ exercise
- Decision rule for prediction?
  Vector product of features and feature weights > 0*?*
- Update for one instance
  - *Make prediction.*
  - *prediction incorrect?*
  - *Depending on the error, correct weights up or down.*

training_iteration(dataset)

- A training cycle on the training data.
- Update is performed for each instance (if misclassified).
- Before each iteration, the data should be remixed.

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

- (Optionally, the number of actual updates can be returned.)

Training

- Several training iterations
- Accuracy is compared on the development data
- At the end, the classifier uses the weights with the best results

```
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

- Save / load an already trained classifier to/from a JSON file.
- What is JSON?
  - Compact data format in an easy-to-read text form
  - Transfer of structured data
  - serialization (saving of objects/structured data)
  - valide JavaScript (*"JavaScript Object Notation"*)
- Example:

```
{
  "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
  }
}
```

Conversion: Python data ⇔ JSON

The content of Python variables can be saved as JSON in a text file.

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

- Content of `testfile.json`:
  `["foo", {"bar": ["baz", null, 1.0, 2]}]`
- JSON read from file:

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

- result of:
  `l == l_reloaded`
  `l is l_reloaded`
  ?

Summary: perceptron

- Misclassification of an instance ...
- ... the feature weights are corrected accordingly
- multiple iterations through training set
- Optimal number of iterations is determined by development data (dev set)