WestlakeNLP

Natural Language Processing

Yue Zhang
Westlake University

西湖大學
WESTLAKE UNIVERSITY

WestlakeNLP

Chapter 3

Feature Vector

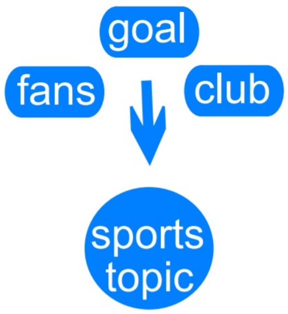
Contents

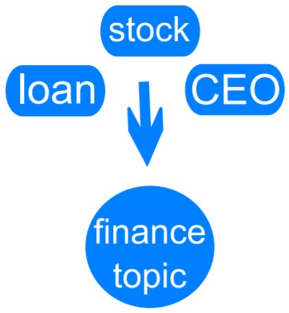
Review Naïve Bayes

$$P(c|d) = P(c) \cdot \prod_{w \in c} P(w|c)$$

$$\log P(c|d) = \log P(c) + \sum_{w \in c} \log P(w|c)$$

Words as features





$$\log P(w|c)$$

Feature vectors

$$\vec{\Phi} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{|V|} \end{bmatrix}$$

$$\vec{\theta}_{sports} = \begin{bmatrix} \log P(goal \mid sports) \\ \log P(fans \mid sports) \\ \vdots \\ \log P(stock \mid sports) \\ \log P(loan \mid sports) \\ \log P(\text{CEO} \mid sports) \end{bmatrix}$$

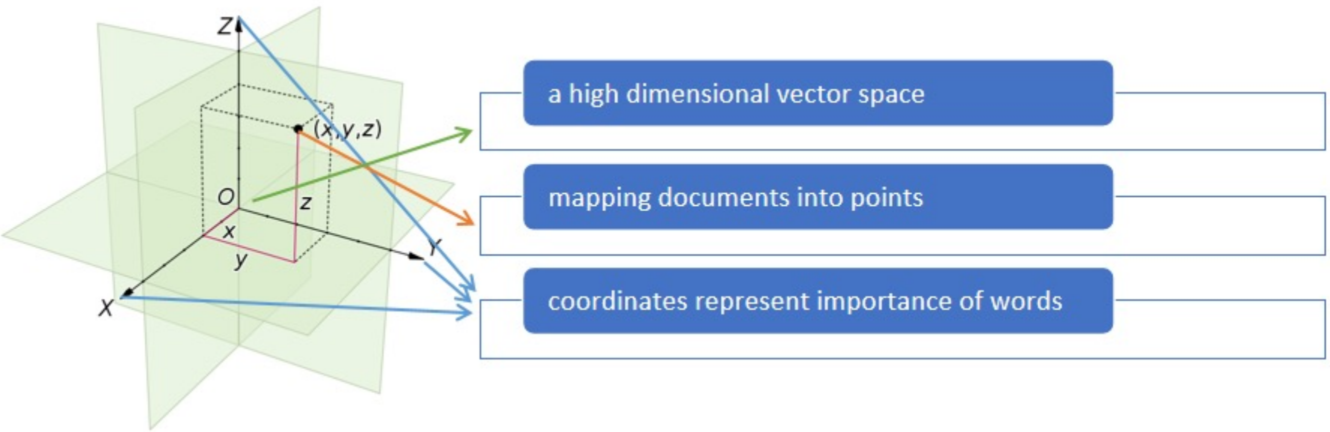
$\mathbf{f}_1$ = # goal ∈ d

$$\log P(c = sports \mid d) = \vec{\theta}_{sports} \cdot \vec{\Phi} + \log P(c = sports)$$

Vector Space Model

Mapping documents to vectors

(unstructured texts into mathematical structures)



Vector representation of documents

d_1 = "Tim bought a book."

d_2 = "Tim is reading a book."

d_3 = "ah, I know Tim."

d_4 = "I saw a boy reading a book."

Features	d_1	d_2	d_3	d_4
w_1 = "a"	1	1	0	2
w_2 = "ah"	0	0	1	0
...				
w_{1001} = "book"	1	1	0	1
w_{2017} = "bought"	1	0	0	0
w_{2100} = "boy"	0	0	0	1
w_{3400} = "I"	0	0	1	1
w_{4400} = "is"	0	1	0	0
...				
w_{5002} = "know"	0	0	1	0
w_{6013} = "reading"	0	1	0	1
w_{7034} = "saw"	0	0	0	1
w_{8400} = "Tim"	1	1	1	0
...				
w_{13200} = ","	0	0	1	0
w_{13201} = "."	1	0	1	0
...				

(a) count-based vectors

Sparse vectors document representation

- Vocabulary: $V = \{w_1, w_2, \ldots, w_{|V|}\}$
- Vector representation for document d :

$$\vec{v}(d) = \langle f_1, f_2, \ldots, f_{|V|} \rangle$$

A simple way to define f but with sparseness :

Count-based vectors (high-dimensional sparse vectors)

$$f_i = \#\mathbf{w}_i \text{ and } \vec{v}(d) = \langle \#\mathbf{w}_1, \#\mathbf{w}_2, \ldots, \#\mathbf{w}_{|V|} \rangle$$

WestlakeNLP

Stop words

- Frequent yet uninformative
- Common stop words in English

| a | the | on | of | with | about | and | in | at | to | " |, |?| oh | . |

- Filter uninformative words

  remove Stop Words from the vocabulary when mapping documents to vectors

- Limitation

  manually defined.

TF-IDF vectors document representation

- Soft version of stop words in selecting useful words.
- Intuition — the more documents in which of words exists, the less informative the word is.
- reduce the importance values of uninformative words

$$\vec{v}_{tf-idf}(d_j) = \langle \frac{TF(w_1, d_j)}{DF(w_1)}, \frac{TF(w_2, d_j)}{DF(w_2)}, \ldots, \frac{TF(w_n, d_j)}{DF(w_n)} \rangle$$

$$= \langle TF(\mathbf{w}_1, d_i) IDF(\mathbf{w}_1), TF(\mathbf{w}_2, d_i) IDF(\mathbf{w}_2),$$

$$\ldots, TF(\mathbf{w}_n, d_i) IDF(\mathbf{w}_n) \rangle$$

TF-IDF vectors

Soft version of stop words in selecting useful words

- Term frequency (TF)

$$TF(\mathbf{w}_i, d_j) = \frac{\#\{\mathbf{w}_i | \mathbf{w}_i \in d_j\}}{\#\{\mathbf{w} | \mathbf{w} \in d_j, \mathbf{w} \in V\}}$$

- Document frequency (DF)

$$DF(\mathbf{w}_i) = \frac{\#\{d | d \in D, \mathbf{w}_i \in d\}}{|D|}$$

- Inverted document frequency (IDF) (with logarithm)

$$IDF(\mathbf{w}_i) = \log \frac{|D|}{\#\{d | d \in D, \mathbf{w}_i \in d\}}$$

WestlakeNLP

Vector representation of documents

d_1 = "Tim bought a book."

d_2 = "Tim is reading a book."

d_3 = "ah, I know Tim."

d_4 = "I saw a boy reading a book."

Features	d_1	d_2	d_3	d_4	d_1	d_2	d_3	d_4
w_1 = "a"	1	1	0	2	0.415	0.415	0	0.83
w_2 = "ah"	0	0	1	0	0	0	2.0	0
...								
w_{1001} = "book"	1	1	0	1	0.415	0.415	0	0.415
w_{2017} = "bought"	1	0	0	0	2.0	0	0	0
w_{2100} = "boy"	0	0	0	1	0	0	0	2.0
w_{3400} = "I"	0	0	1	1	0	0	1.0	1.0
w_{4400} = "is"	0	1	0	0	0	2.0	0	0
...								
w_{5002} = "know"	0	0	1	0	0	0	2.0	0
w_{6013} = "reading"	0	1	0	1	0	1.0	0	1.0
w_{7034} = "saw"	0	0	0	1	0	0	0	2.0
w_{8400} = "Tim"	1	1	1	0	0.415	0.415	0.415	0
...								
w_{13200} = ","	0	0	1	0	0	0	2.0	0
w_{13201} = "."	1	0	1	0	1.0	0	1.0	0
...								

(a) count-based vectors (b) TF-IDF vectors

Summary

Vector representation for document d

$$\vec{v}(d) = \langle f_1, f_2, \ldots, f_{|V|} \rangle$$

- In count-based vectors, $f_i = \#w_i = TF(w_i, d_j)$
- In TF-IDF vectors, $f_i = \frac{TF(w_i, d_j)}{DF(w_i)}$

Feature extraction

- Mathematical abstraction: the process of transforming document d into vector $\vec{\boldsymbol{v}}(d)$
- Count-based vectors: discrete features
- TF-IDF vectors: real-valued features

Hands on

WestlakeNLP

A case study on a tiny corpora

d_1: Tim bought a book.

d_2 : Tim is reading a book.

d_3 : ah, Tim is Tim.

d_4 : I saw a boy reading a book.

- Create an index vocabulary of the words of the train document set:

$$V = \begin{bmatrix} w_1 = Tim \\ w_2 = bought \\ w_3 = book \\ w_4 = reading \\ \dots \end{bmatrix}$$

* Certain stop words were ignored

Hands on

WestlakeNLP

Python practice 1

Count-based document representation.

1. Import python modules pytorch , collections and math

```
import torch
from collections import Counter
import math
```

2. Load dataset and define the stop-words

```
documents = ["Tim bought a book .",
             "Tim is reading a book .",
             "ah , Tim is Tim .",
             "I saw a boy reading a book ."]

stop_words = ['a', '.', ',']
```

Hands on

WestlakeNLP

3. Clean stop-words and count word frequency

```
clean_docs = []
word_count = Counter()
for doc in documents:
    word_count.update([wd for wd in doc.strip().split(' ')
                       if wd not in stop_words])
    clean_docs.append([wd for wd in doc.strip().split(' ')
                       if wd not in stop_words])
```

4. Build up the vocabulary

```
vocab = [word for word in word_count.keys()]
```

Hands on

WestlakeNLP

Check the loaded data

```
print(clean_docs)

[['Tim', 'bought', 'book'],
['Tim', 'is', 'reading', 'book'],
['ah', 'Tim', 'is', 'Tim'],
['I', 'saw', 'boy', 'reading', 'book']]
```

Word count

```
print(word_count)

Counter({'Tim': 4, 'book': 3, 'is': 2, 'reading': 2,
         'bought': 1, 'ah': 1, 'I': 1, 'saw': 1, 'boy': 1})
```

Vocabulary

```
print(vocab)

['Tim', 'bought', 'book', 'is',
'reading', 'ah', 'I', 'saw', 'boy']
```

Hands on

WestlakeNLP

6. Count-based document representation

```
count_vec = torch.zeros(len(clean_docs), len(vocab))
for i in range(len(clean_docs)):
    for j in range(len(vocab)):
        count = 0
        for word in clean_docs[i]:
            if word == vocab[j]:
                count += 1
        count_vec[i][j] = count
```

Result:

```
print(count_vec)

tensor([[1., 1., 1., 0., 0., 0., 0., 0., 0.],
        [1., 0., 1., 1., 1., 0., 0., 0., 0.],
        [2., 0., 0., 1., 0., 1., 0., 0., 0.],
        [0., 0., 1., 0., 1., 0., 1., 1., 1.]])
```

Is there a soft alternative ?

Hands on

WestlakeNLP

Python practice 2

TF-IDF vectors calculation using python.

7. Count the number of documents that contain a certain vocabulary word

```
doc_count = torch.ones(1, len(vocab))

for i in range(len(vocab)):
    freq = 0
    for doc in clean_docs:
        if vocab[i] in doc:
            freq += 1
    doc_count[0][i] = freq
```

```
print(doc_count)

tensor([[3., 1., 3., 2., 2., 1., 1., 1., 1.]])
```

* Problem set succeeded from python practice 1

Hands on

8. Count the vocabulary words in each document

```
doc_len = torch.zeros(len(clean_docs), 1)
for i in range(len(clean_docs)):
    doc_len[i][0] = len(clean_docs[i])
```

9. Calculate the term frequency

```
tf = count_vec/doc_len
```

Result:

```
print(tf)

tensor([[0.3333, 0.3333, 0.3333, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.2500, 0.0000, 0.2500, 0.2500, 0.2500,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.5000, 0.0000, 0.0000, 0.2500, 0.0000,
         0.2500, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.2000, 0.0000, 0.2000,
         0.0000, 0.2000, 0.2000, 0.2000]])
```

Hands on

10. Calculate the inverted document frequency

```
idf = torch.log(torch.ones(len(documents),
                len(vocab))*len(documents)/doc_count)
```

Result:

```
print(idf)

tensor([[0.2877, 1.3863, 0.2877, 0.6931, 0.6931,
         1.3863, 1.3863, 1.3863, 1.3863],
        [0.2877, 1.3863, 0.2877, 0.6931, 0.6931,
         1.3863, 1.3863, 1.3863, 1.3863],
        [0.2877, 1.3863, 0.2877, 0.6931, 0.6931,
         1.3863, 1.3863, 1.3863, 1.3863],
        [0.2877, 1.3863, 0.2877, 0.6931, 0.6931,
         1.3863, 1.3863, 1.3863, 1.3863]])
```

Hands on

WestlakeNLP

11. TF-IDF vector document representation

```
tfidf = tf*idf
```

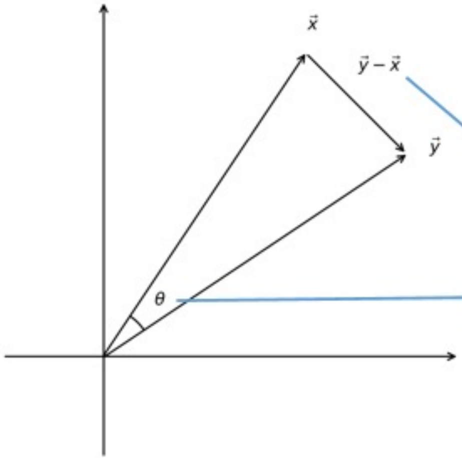
```
print(tfidf)

tensor([[0.0959, 0.4621, 0.0959, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0719, 0.0000, 0.0719, 0.1733, 0.1733,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.1438, 0.0000, 0.0000, 0.1733, 0.0000,
         0.3466, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0575, 0.0000, 0.1386,
         0.0000, 0.2773, 0.2773, 0.2773]])
```

Contents

WestlakeNLP

Measure vector space distance

unstructed texts and NLP structures ⇒ vector space distance ⇒ semantic similarities



Euclidean distance

- measures the length of their difference |y˜-x˜|

Cosine similarity

- measure the size of the angel θ

Measure vector space distance

WestlakeNLP

$\vec{X} = \langle x_1, x_2, \ldots, x_n \rangle$ $\vec{Y} = \langle y_1, y_2, \ldots, y_n \rangle$

- Euclidean distance

$$dis^{\text{eu}}(\vec{x}, \vec{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

- Cosine distance

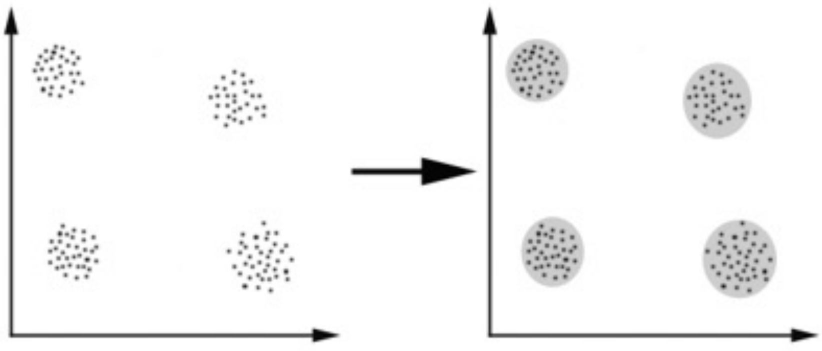
$$dis^{\cos}(\vec{x}, \vec{y}) = 1 - \cos(\vec{x}, \vec{y})$$

From cosine similarity

$$\cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}||\vec{y}|} = \frac{x_1 y_1 + x_2 y_2 + \cdots + x_n y_n}{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}\sqrt{y_1^2 + y_2^2 + \cdots + y_n^2}}$$

Clustering

To find groups of vectors that stay relatively close to each other, using measures of distance in vector space (Euclidean distance as the metric)



Contents

WestlakeNLP

K-means clustering

Iteratively assigns points to clusters based on their distance to the centroids

Initialization: pre-specify the number of clusters k
randomly select k points as cluster centroids

Steps:
repeat:
a. assign each point to the cluster whose centroid is the closest;
b. reassign cluster centroids (by averaging points in each cluster);
until:
the cluster contents stabilize

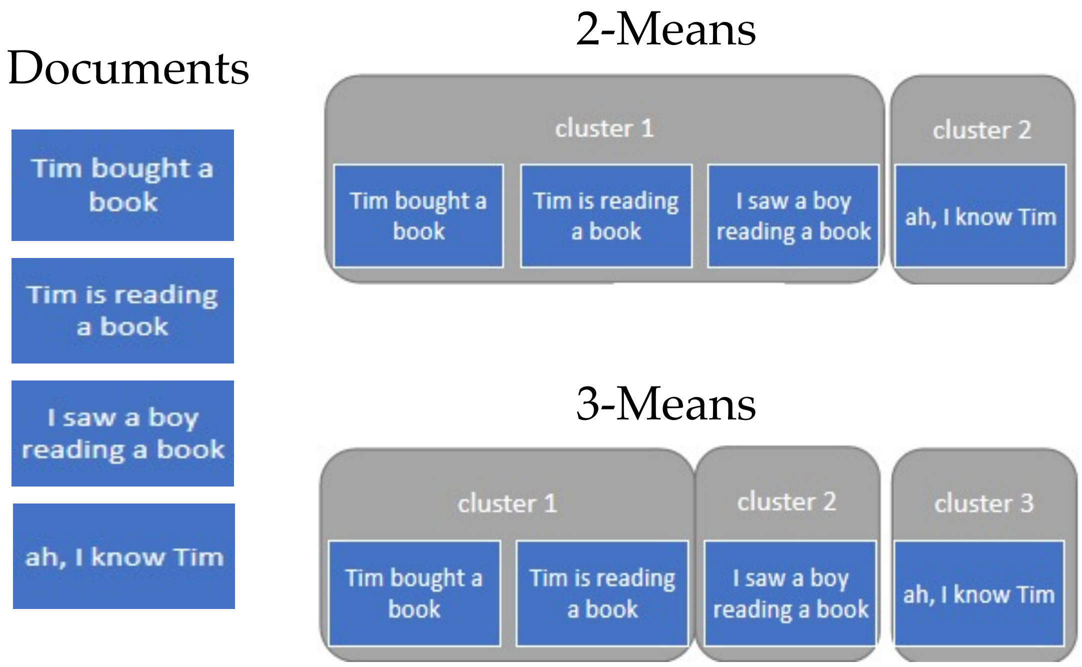
K-means clustering

WestlakeNLP

Algorithm 1: K-means.

```
Inputs: D = {v⃗_1, v⃗_2, ..., v⃗_N}, K;
Initialization: clusters = [], centroids = []
for k ∈ [1...K] do
    clusters.APPEND([]);
    centroids.APPEND(D[RANDOM(j ∈ [1...N] and j ∉ centroids)]);
repeat
    clusters_old ← clusters;
    clusters ← [];
    // assign points to clusters
    for i ∈ [1...N] do
        c_j ← arg min_j DIST(D[i], centroids[j]);
        clusters[c_j].APPEND(D[i]);
    // calculate centroids
    for k ∈ [1...K] do
        centroids[k] ← AVERAGE(clusters[k]);
until clusters = clusters_old;
Outputs: clusters
```

K-means clustering



Hands on

WestlakeNLP

Python practice 3

Calculate Euclidean distance and cosine distance using pytorch

For our documents $[d_1, d_2, d_3, d_4]$, calculate their similarity using torch.dist and torch.cosine_similarity

Compare the distance between d_1 and d_2 , to the distance between d_3 and d_4, what you can see?

1. Assign the TF-IDF vector representation to the target documents

```
...
d1 = tfidf[0]
d2 = tfidf[1]
d4 = tfidf[3]
```

* Problem set succeeded from python practice 1

Hands on

WestlakeNLP

2. Calculate Euclidean distance using the pytorch module

```
d1_d2 = torch.dist(d1, d2)
d1_d4 = torch.dist(d1, d4)
```

Result:

```
print(d1_d2, d1_d4)

tensor(0.5242) tensor(0.6885)
```

3. Calculate cosine distance using the pytorch module

```
d1_d2 = torch.cosine_similarity(d1, d2, dim=0)
d1_d4 = torch.cosine_similarity(d1, d4, dim=0)
```

Result:

```
print(d1_d2, d1_d4)

tensor(0.1079) tensor(0.0228)
```

Hands on

WestlakeNLP

K-means clustering with python

2-means and 3-means clustering using Scikit.learn

```
from sklearn.cluster import KMeans
km_cluster = KMeans(n_clusters=2,max_iter=300,n_init=40,
                    init="k-means++",n_jobs=-1)
result = km_cluster.fit_predict(tfidf_matrix)
print("Predicting result for 2-means:",result)
Predicting result for 2-means: [1 1 0 1]

km_cluster = KMeans(n_clusters=3,max_iter=300,n_init=40,
                    init="k-means++",n_jobs=-1)
result = km_cluster.fit_predict(tfidf_matrix)
print("Predicting result for 3-means:",result)
Predicting result for 3-means: [0 0 1 2]
```

init : method for initialization, defaults to "k-means++".

n_init : number of time the k-means algorithm will be run with different centroid seeds.

n_jobs : the number of jobs to use for the computation.

Contents

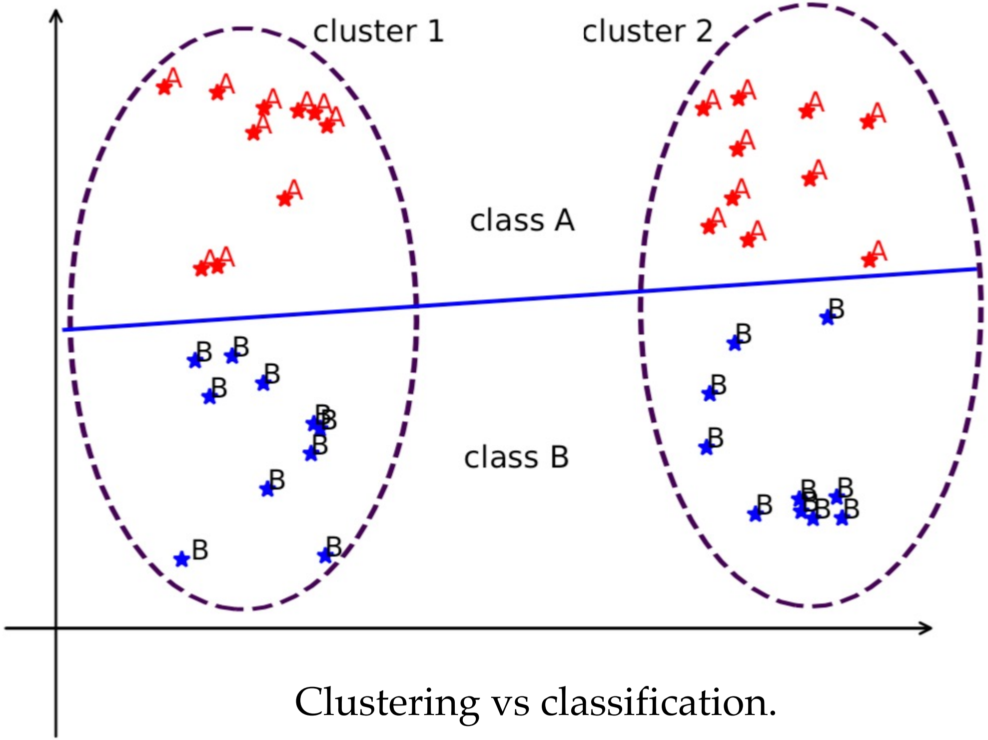
WestlakeNLP

Clustering vs. classification

My emails

	Travel	Non-travel
Work	d_1 d_2 d_3	d_4 d_5 d_6 d_7
Leisure	d_8 d_9 d_{10} d_{11}	d_{12} d_{13} d_{14} d_{15} d_{16}

Clustering vs. classification



Clustering vs classification.

Clustering vs. classification

$$\vec{\Phi} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{|V|} \end{bmatrix}$$

$$\vec{\theta}_{sports} = \begin{bmatrix} \log P(goal \mid sports) \\ \log P(fans \mid sports) \\ \vdots \\ \log P(stock \mid sports) \\ \log P(loan \mid sports) \\ \log P(\text{CEO} \mid sports) \end{bmatrix}$$

$\mathbf{f}_1$ = # goal ∈ d

$$\log P(c = sports \mid d) = \vec{\theta}_{sports} \cdot \vec{\Phi} + \log P(c = sports)$$

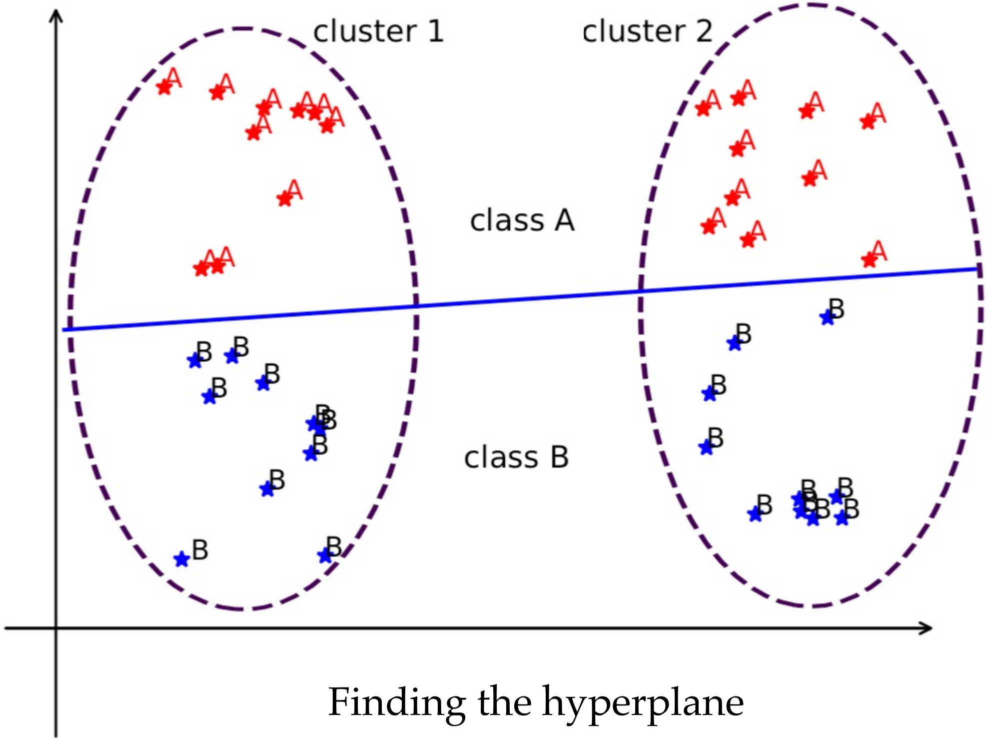
Clustering vs. classification

WestlakeNLP

- Clustering (unsupervised learning)
  - Do not require manually labeled training data
  - All words have equal importance in a document vector
  - Difficult to ensure customized vector division
- Classification (supervised learning)
  - Requires training data with manual class labels
  - Pick up the important words for classification tasks
  - Use model parameters to define space separation

Contents

Vector space classification task



Finding the hyperplane

Linear separability

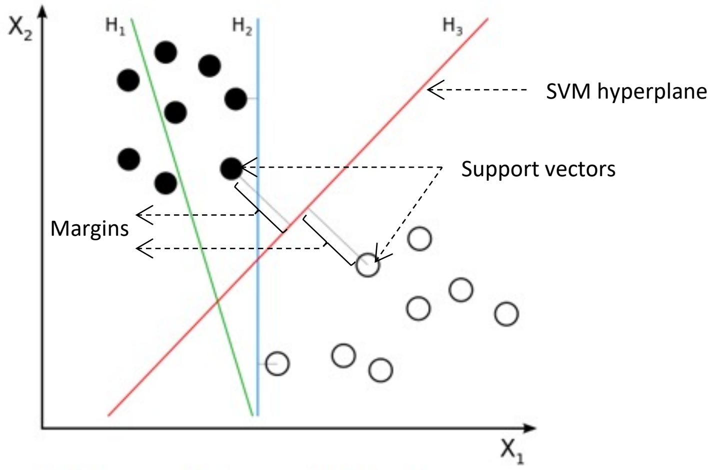
WestlakeNLP

- **Hyperplane**: linear shape in a high-dimensional vector space.
  - 2-dimensional space: line
  - 3-dimensional space: plane
  - dimension $\geq$ 3: hyperplane
- **Linear separable**: labeled points have a **hyperplane** separation boundary
- **Linear models** : a balance between accuracy and complexity
  - support vector machine
  - perceptron algorithm

Support vector machine (SVM)

WestlakeNLP

- Definition: a linear model for binary classification in vector space
- Support vectors: points closest to the separating hyperplane
- Margins: Support vector distances to the hyperplane
- Training goal: find the hyperplane that maximizes the margins



H1 does not separate the classes.
H2 does, but only with a small margin.
H3 separates them with the maximum margin.

SVM classifier

- Defining the hyperplane

$$\vec{w}^T\vec{v} + b = 0$$

  - $\vec{w}$ is a normal vector perpendicular to the hyperplane
  - On one side, $\vec{w}^T\vec{v} + b > 0$; on the other side, $\vec{w}^T\vec{v} + b < 0$
  - Distance between vectors to the hyperplane:

$$r = \frac{|\vec{w}^T\vec{v}(x) + b|}{||\vec{w}||}$$

Parameterising the model

- There is an infinite amount of $(\overrightarrow{w}, b)$ pairs to define each hyperplane
- For one unique $(\overrightarrow{w}, b)$ pair, SVM chooses the scale according to training data, requiring that $\left|\vec{\omega}^T \vec{v}\left(x_s\right)+b\right|=1$ for all support vectors $\vec{v}(x_s)$
- For any support vector $\vec{v}(x_s)$ in the set of training examples, its distance to the separating hyperplane is

$$r=\frac{\left|\vec{\omega}^T \vec{v}\left(x_i\right)+b\right|}{\|\vec{\omega}\|}=\frac{1}{\|\overrightarrow{w}\|}$$

SVM classifier

- **Finding the hyperplane**

  The goal of SVM training is to find a hyperplane $\vec{w}^T\vec{v} + b = 0$

  that maximizes $2r = \frac{2}{||\vec{w}||}$,

  such that *y=+1*/*y=-1* resides on different sides of the hyperplane.

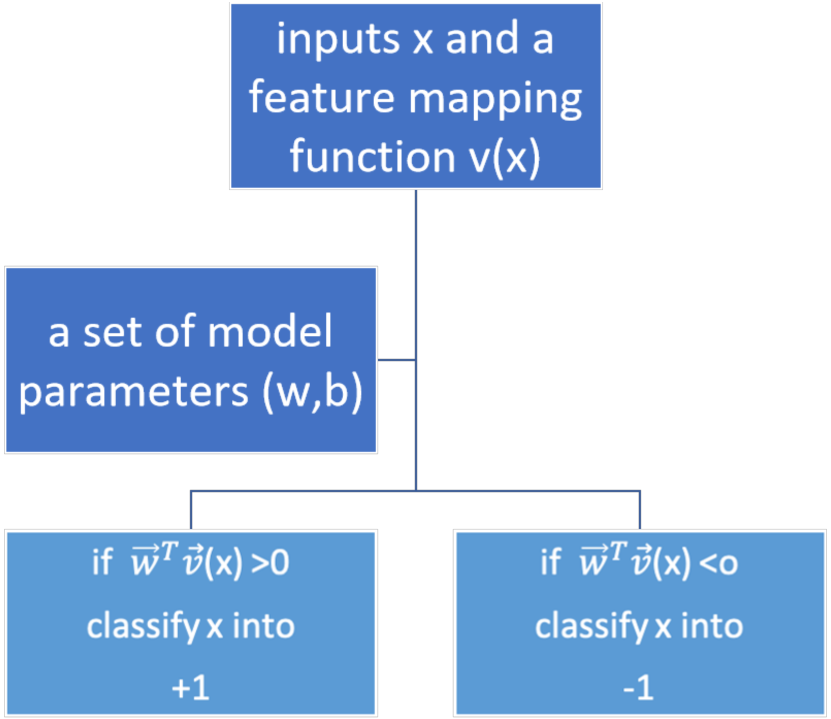
- **Equivalent** to minimizing $\frac{1}{2}||\vec{w}||^2$

- **Training objective**

$$(\vec{w}_{sum}, b_{sum}) = \arg\min \frac{1}{2}||\vec{w}||^2,$$

$$s.t.\quad y_i(\vec{w}^T\vec{v}(x_i) + b) \geq 1, \text{for all } (x_i, y_i) \in D$$

Test scenarios



Contents

WestlakeNLP

The perceptron algorithm

- a linear model to find a value for $(\vec{w}, b)$

  such that y = SIGN $(\vec{w}^T \vec{v}(x_i) + b)$ for all training examples (x_i, y_i)

Initialization: set $\vec{w}$ to $\vec{0}$, b to 0

Steps:

repeat:

for each input x calculate a current output z

if the output y is different from the gold output z :

Adjust the model parameter $\vec{w}$ by

adding $\vec{v}(x)$ if y = +1

subtracting $\vec{v}(x)$ if y = -1

Adjust b by

adding 1if y = +1

subtracting 1 if y = -1

until:

a certain iteration number is reached.

The perceptron algorithm

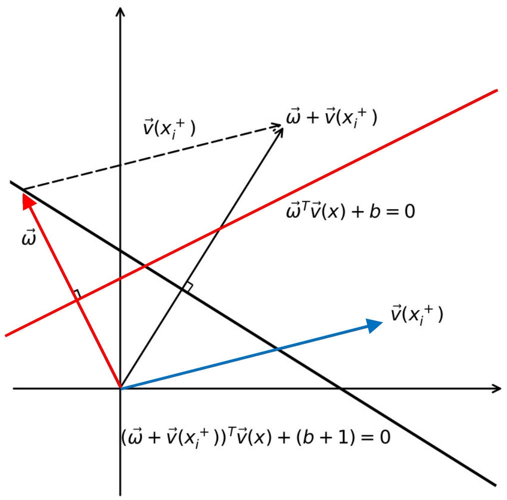
WestlakeNLP

- Algorithm

Input: $D = \{(x_i, y_i)\}|_{i=1}^{N}, y_i \in \{-1, +1\}$
Initialization: $\vec{\omega} \leftarrow \vec{0}; b \leftarrow 0; t \leftarrow 0$
repeat
  for $i \in [1..N]$ **do**
    $z_i \leftarrow \textsc{Sign}(\vec{\omega}^T \vec{v}(x_i) + b);$
    if $z_i \neq y_i$ **then**
      $\vec{\omega} \leftarrow \vec{\omega} + \vec{v}(x_i) \times y_i;$
      $b \leftarrow b + y_i;$
  $t \leftarrow t + 1;$
until $t = T;$

Perceptron update

- vector space interpretation



If the correct training example $\vec{v}(x_i^+)$ falls on the wrong side of the hyperplane, the perceptron update changes the normal vector $\vec{w}$ towards $\vec{v}(x_i^+)$, and changes b by 1.

Numerical Interpretation

- Given a model $(\vec{\omega}, b)$.
- The current instance x_i^+has $\vec{\omega}^T x_i^+ + b < 0$.
- The new model becomes $(\vec{\omega} + \vec{v}(x_i^+), b + 1)$ after the update.
- The new score is

  $\left(\vec{\omega} + \vec{v}(x_i^+)\right)^T \vec{v}(x_i^+) + b + 1 = (\vec{\omega}^T \vec{v}(x_i^+) + b) + \left(\vec{v}(x_i^+)\right)^2 + 1$, which

  is larger than the old score $\vec{\omega}^T \vec{v}(x_i^+) + b$.
- Thus x_i^+ will be more likely on the positive side of the new hyperplane.

Batch learning vs online learning

- Batch learning algorithm

  SVM defines a training objective over a full set of training data

- Online learning algorithm

  Perceptron updates its parameters incrementally for each training example

Contents

Solutions towards multi-class classification

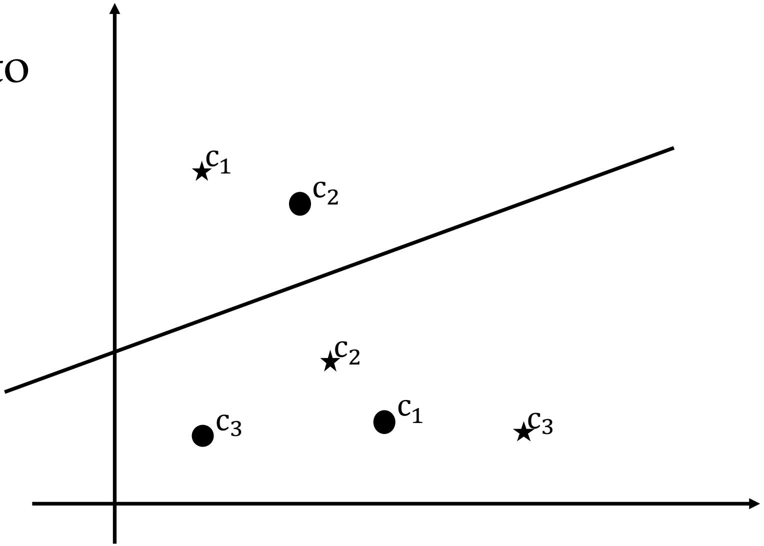
- One-vs-rest approaches
  - a hyperplane separates out a particular class of document from the rest
  - Pairwise approaches
- More principled solutions
  - One linear model
  - Two views
    - vector space separation
    - scoring function

Contents

WestlakeNLP

Solutions towards multi-class classification

- Vector space is separated into correct output and incorrect output subspaces.
- the ratio between the numbers of positive and negative examples is constantly (1 : |C| - 1).



Multi-class classification (★ and ● are two documents, c_1 , c_2 and c_3 are three class labels. The gold label for ★ is c_1 and the gold label for ● is c_2.)

Output-based features

Input-based feature vector : $\vec{v}(x)$

$\Downarrow$

Output-based feature vector : $\vec{v}(x, \mathbf{c})$

NOTE : x – input, c – class

Cartesian product (count-based vector $\vec{v}(d)$ for example)

$$\vec{v}(d) = \langle \#w_1, w_2, \ldots, w_{|v|} \rangle$$

$$\begin{aligned}\vec{v}(d, \mathbf{c}) = \langle & \#\mathbf{w}_1\mathbf{c}_1, \#\mathbf{w}_2\mathbf{c}_1, \ldots, \#\mathbf{w}_{|V|}\mathbf{c}_1 \\ & \#\mathbf{w}_1\mathbf{c}_2, \#\mathbf{w}_2\mathbf{c}_2, \ldots, \#\mathbf{w}_{|V|}\mathbf{c}_2 \\ & \ldots \\ & \#\mathbf{w}_{|V|}\mathbf{c}_{|C|}, \#\mathbf{w}_{|V|}\mathbf{c}_{|C|}, \ldots, \#\mathbf{w}_{|V|}\mathbf{c}_{|C|} \rangle\end{aligned}$$

Output-based features

- Document: Tim went to Amsterdam to meet Jason
- Label: Work
- Output-based features:

Tim|Work	went|Work	to|Work
1	1	2
Amsterdam|Work	meet|Work	Jason|Work
1	1	1

Contents

WestlakeNLP

Multi-class SVM

- Training examples: $D = \{(x_i, c_i)\}_{i=1}^{N}$,
- Positive examples: $\vec{v}(x_i, c_i)$
- Negative examples : $\vec{v}(x_i, \boldsymbol{c})$, where $\boldsymbol{c} \neq c_i$
- Training objective:

$$\hat{\omega}, \hat{b} = \arg\min_{\vec{w}, b} \frac{1}{2} ||\vec{\omega}||^2$$

$$\text{s.t. for all } i, x_i \in D \begin{cases} \vec{\omega}^T \vec{v}\,(x_i, c_i) + b \geq 1 \\ \text{for all } \mathbf{c} \neq c_i, \vec{\omega}^T \vec{v}\,(x_i, \mathbf{c}) + b \leq -1 \end{cases}$$

- Test time find the class as $\arg\max_{\mathbf{c} \in C} \vec{\omega}^T \vec{v}(x, \mathbf{c}) + b$

Multi-class SVM

WestlakeNLP

- Training examples: $D = \{(x_i, c_i)\}_{i=1}^{N}$,
- Positive examples: $\vec{v}(x_i, c_i)$
- Negative examples : $\vec{v}(x_i, \boldsymbol{c})$, where $\boldsymbol{c} \neq c_i$
- Training objective:

$$\hat{\omega}, \hat{b} = \arg\min_{\vec{\omega}, b} \frac{1}{2} \|\vec{\omega}\|^2$$

Too strict?

$$\text{s.t. for all } i, x_i \in D \begin{cases} \vec{\omega}^T \vec{v}(x_i, c_i) + b \geq 1 \\ \text{for all } \mathbf{c} \neq c_i, \vec{\omega}^T \vec{v}(x_i, \mathbf{c}) + b \leq -1 \end{cases}$$

- Test time find the class as $\arg\max_{\mathbf{c} \in C} \vec{\omega}^T \vec{v}(x, \mathbf{c}) + b$

Linear models as scoring functions

- In a score perspective:

$$score(x, \mathbf{c}) = \vec{\omega}^T \vec{v}(x, \mathbf{c}) + b$$

Given a test input *x*, the model finds the class label ĉ with the highest score as the output:

$$\hat{c} = \arg\max_{\mathbf{c} \in C} score(x, \mathbf{c}) = \arg\max_{\mathbf{c} \in C} \vec{\omega}^T \vec{v}(x, \mathbf{c}) + b$$

- Final form of multi-class SVM training objective

$$\hat{\omega} = \arg\min \frac{1}{2} ||\vec{\omega}||^2$$
$$\text{s.t. } \vec{\omega}^T \vec{v}(x_i, c_i) - \vec{\omega}^T \vec{v}(x_i, \mathbf{c}) \geq 1 \text{ for all } \mathbf{c} \neq c_i$$

Contents

Multi-class perceptron

- Algorithm 3.3

Input: $D = (x_i, c_i)|_{i=1}^{N}$, $c_i \in C$
Initialization: $\vec{\omega} \leftarrow \vec{0}$; $t \leftarrow 0$;
repeat
  for $i \in [1 \ldots N]$ **do**
    $z_i \leftarrow \arg\max_{\mathbf{z}} \vec{\omega}^T \vec{v}(x_i, \mathbf{z})$;
    if $z_i \neq c_i$ **then**
      $\vec{\omega} \leftarrow \vec{\omega} + \vec{v}(x_i, c_i) - \vec{v}(x_i, z_i)$;
  $t \leftarrow t + 1$;
until $t = T$;

- Goes through training examples multiple iterations
- update parameter vector by adding the feature vector of the correct output and abstracting the feature vector of the incorrect prediction

Multi-class perceptron

- Given a model $\vec{\omega}$,
- The current instance x_i^+ has $\vec{\omega} \cdot \vec{v}\,(x_i, z) > \vec{\omega} \cdot \vec{v}\,(x_i, c_i)$
- The new model parameters become $\vec{\omega} + \vec{v}\,(x_i, c_i) - \vec{v}\,(x_i, z)$ after the update.
- The new score difference

$$\left(\vec{\omega} + \vec{v}\,(x_i, c_i) - \vec{v}\,(x_i, z)\right) \cdot v(x_i, z) - \left(\vec{\omega} + \vec{v}\,(x_i, c_i) - \vec{v}\,(x_i, z)\right) \cdot v(x_i, c_i) =$$

$$\vec{\omega}\left(\vec{v}\,(x_i, z) - \vec{v}\,(x_i, c_i)\right) - (\vec{v}\,(x_i, z) - \vec{v}\,(x_i, c_i))^2 < \vec{\omega}\left(\vec{v}\,(x_i, z) - \vec{v}\,(x_i, c_i)\right)$$

- More likely being correct.

Contents

Descriminative models

- Both SVM and perceptron are discriminative models
  They work by differentiating positive examples and negative examples,
  (for binary classification y=+1/y=-1; for multi-class classification c)
  assigning higher scores to positive examples
- Naïve Bayes is a generative model, calculating joint probabilities of inputs and outputs
- All the three models are linear models

Naïve Bayes is a Linear Model too

$$\vec{\Phi} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{|V|} \end{bmatrix}$$

$$\vec{\theta}_{sports} = \begin{bmatrix} \log P(goal \mid sports) \\ \log P(fans \mid sports) \\ \vdots \\ \log P(stock \mid sports) \\ \log P(loan \mid sports) \\ \log P(\text{CEO} \mid sports) \end{bmatrix}$$

$\mathbf{f}_1$ = # goal $\in d$

$$\log P(c = sports \mid d) = \vec{\theta}_{sports} \cdot \vec{\Phi} + \log P(c = sports)$$

Naïve Bayes is a Linear Model too

$$\vec{\Phi} = \begin{bmatrix} \mathbf{1} \\ \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{|V|} \end{bmatrix}$$

$$\vec{\theta}_{sports} = \begin{bmatrix} \log P(sports) \\ \log P(goal \mid sports) \\ \log P(fans \mid sports) \\ \vdots \\ \log P(stock \mid sports) \\ \log P(loan \mid sports) \\ \log P(\mathrm{CEO} \mid sports) \end{bmatrix}$$

$\mathbf{f}_1$ = # goal ∈ d f_0=**1**

$$\log P(c = sports \mid d) = \vec{\theta}_{sports} \cdot \vec{\Phi}$$

Discriminative model vs. generative model

Generative models	Discriminative models
Naïve Bayes classifier	SVMs
	Perceptron

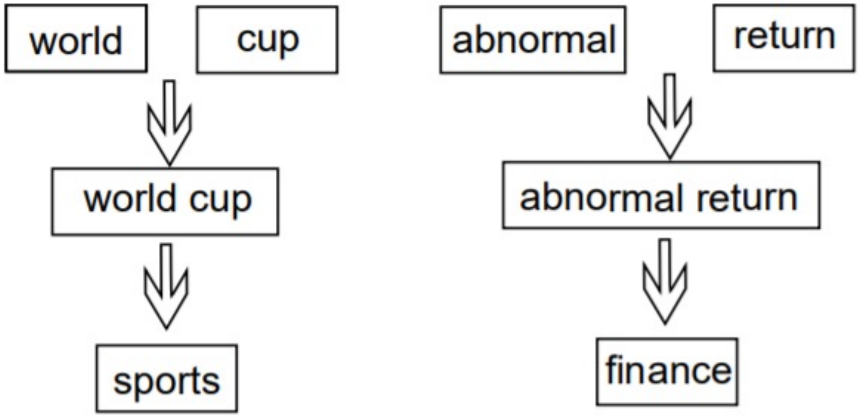
- Parameter types P(c), P(w | c) and parameter instances P(sports), P(goal | sports)
- Feature vectors are assembly of parameter instances. But we can add more parameter types into our feature vectors

$$\vec{v}(d,c) = \langle w_1 c, w_2 c, \dots, w_{|V|} c \rangle \Longrightarrow \vec{v}(d,c) = \langle c_1, c_2, \dots, c_{|C|}, w_1 c, w_2 c, \dots, w_{|V|} c \rangle$$

- Advantage of discriminative models:

  using overlapping features, such as word and bigram features.

Bigram features



- Bigram features are useful for text classification, they offer more specific information about text classes
- Bigrams are sparser making the feature vector longer and more sparse

Add bigram features

WestlakeNLP

$$\vec{v}(d) = \langle \mathbf{w}_1, \mathbf{w}_2 \ \ldots, \mathbf{w}_{|V|}, \mathbf{bi}_1, \mathbf{bi}_2, \ldots, \mathbf{bi}_{|BI|} \rangle$$

As in the example from textbook, with bigram features, the feature vector for the sentence "Tim bought a book" in Table 3.1 is

$$\langle f_1 = \mathbf{w}_1 = 1, f_2 = \mathbf{w}_2 = 0, \ldots, f_{1001} = \mathbf{w}_{1000} = 1, \ldots, f_{2017} = \mathbf{w}_{2017} = 1, \ldots, f_{8400} = \mathbf{w}_{8400} = 1, \ldots, f_{13201} = \mathbf{w}_{13201} = 1, \ldots, f_{|V|+1} = \mathbf{bi}_1 = 0, \ldots, f_{|V|+108} = \mathbf{bi}_{108} = 1, \ldots, f_{|V|+3650} = \mathbf{bi}_{3650} = 1, \ldots, f_{|V|+4950} = \mathbf{bi}_{4950} = 1, \ldots, f_{|V|+113525} = \mathbf{bi}_{113525} = 1, \ldots \rangle$$

> $\mathbf{bi}_{108}$: a book
> $\mathbf{bi}_{3650}$: book .
> $\mathbf{bi}_{4950}$: bought a
> $\mathbf{bi}_{113525}$: Tom bought

$$\vec{v}(d,c) = \left\langle c_1, c_2, \ldots, c_{|C|}, w_1 c, w_2 c, \ldots, w_{|V|} c, b_{i_1} c, b_{i_2} c, \ldots, b_{i_{|B|}} c \right\rangle$$

Feature templates and instances

WestlakeNLP

- Feature extraction:

    A process of matching feature templates to output structures and instantiating them into feature instances.

- Feature templates: similar to parameter type; in examples above, there are three templates, namely c, wc and $w_{i-1}w_ic$
- Feature instances: similar to parameter instance; e.g., c_2, w_1c_1

Contents

Dot-product form of linear models

A general form of a linear model :

- Given an input x, its score is computed by

Parameter vector (weight vector)

$$score(x, \mathbf{c}) = \theta \cdot \phi(x, \mathbf{c})$$

Feature vector

Dot-product form of linear models

A general form of a linear model :

- Given an input x, its score is computed by

Parameter vector (weight vector)

$$score(x, \mathbf{c}) = \theta \cdot \phi(x, \mathbf{c})$$

Feature vector

- Effectively same as having $score(x, c) = \overrightarrow{\theta_c} \cdot \vec{\phi}(x)$.

Contents

Separability and generalizability

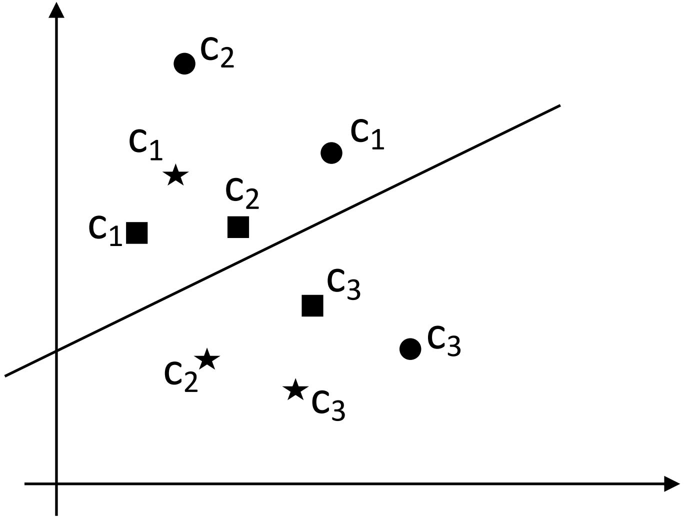
WestlakeNLP

- Feature engineering : the process of defining a useful set of features
  - more feature reflect richer information
  - better designed feature vectors allow better linear separability
- Separability
  - linear separable
  - dataset can be largely linear separable given proper feature definitions
- Generalization
  - overfitting
  - underfitting

Contents

WestlakeNLP

Non-linearly-separable data



Multi-class classification (★ , ■and ● are three documents, c_1 , c_2 and c_3 are three class labels. The gold label for ★ is c_1 and the gold label for ● is c_2, and The gold label for ■ is c_3)

Binary SVM

- Slack variables ξ

$$y\left(\vec{\omega}^T\vec{v}(x)+b\right)=1-\xi \text{ for all } (x_i,y_i)$$

- Training objective

$$(\vec{\omega},b)=\arg\min_{(\vec{\omega},b)} C\sum_i \xi_i+\frac{1}{2}||\vec{\omega}||^2$$
$$\text{s.t. for all } i, y_i\left(\vec{\omega}^T\vec{v}(x_i)+b\right)=1-\xi_i, \xi_i\geq 0$$

$$(\vec{w},b)=\arg\min_{(\vec{w},b)} C\sum_i \max\left(0,1-y_i(\vec{w}^T\vec{v}(x_i)+b)\right)+\frac{1}{2}\left||\vec{w}|\right|^2$$

Multi-class SVM

- Training objective with slack variables :

$$\hat{\vec{\theta}} = \arg\min_{\vec{\theta}} \frac{1}{2}||\vec{\theta}||^2 + C\left(\sum_{i=1}^{N} \xi_i\right)$$

s.t. for all $(x_i, c_i) \in D$:

$$\vec{\theta} \cdot \vec{\phi}(x_i, c_i) = \vec{\theta} \cdot \vec{\phi}(x_i, \mathbf{c}) + 1 - \xi_i, \mathbf{c} \neq c_i, \xi_i \geq 0$$

$$\hat{\vec{\theta}} = \arg\min_{\vec{\theta}} \frac{1}{2} \left||\vec{\theta}|\right|^2 + C(\sum_{i=1}^{N} \max(0,1 - \vec{\theta} \cdot \vec{\phi}(x_i, c_i) + \max_{c \neq c_i}(\vec{\theta} \cdot \vec{\phi}(x_i, c\,))\,)$$

Perceptron

In the case where the training data are not linearly separable, the perceptron can still converge to a model that gives reasonably small numbers of training errors

Summary

- Vector representations of documents
- Support vector machine and perceptron algorithms for binary text classification
- Feature representations of input-output pairs
- Multi-class SVMs and perceptions
- Discriminative models vs generative models
- The importance of features to the separability of training data and generalization to test data