

CSCI 544

Applied Natural Language Processing

Mohammad Rostami
USC Computer Science Department



Logistical Notes

- HW1 is out: start working early
- NLTK
- sklearn
- Quiz1: Blackboard
- Recorded lectures: blackboard -> tools -> usc zoom pro meeting -> cloud recordings.
- Slack Channel: forming groups (necessary for projects and very helpful for doing assignments)
- TA office hours: ask their advice for picking your topic
- Happy to see some people started to form groups and already learning using NLTK

Classification

- Categorizing instances of data into “classes”, where class members share some notion of similarity, e.g., having positive sentiment

Parametric Model:

Learning \approx Choosing and selecting the **best** model

$$y_i = h(\theta) \approx \theta^\top x_i, \theta \in \mathbb{R}^d$$

Model Selection:

$$\hat{\theta} = \arg \min_{\theta \in \Omega(\theta)} \underbrace{\sum_i \mathcal{L}(\theta^\top x_i, y_i)}_{\text{Empirical Error}}$$



Class 0

Class 1

$$\mathbf{X} = [x_1, \dots, x_N] \in \mathbb{R}^{d \times N}, \mathbf{Y} \in \{0, 1\}^N$$

Training Dataset

Features!

Preprocessing

Tokenization: splitting the text into units for processing

- Removing extra spaces: the quality is high
- Removing stop words: article, propositions, etc.

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

- Contractions are standardized: won't -> will not
- Removing unhelpful words, e.g., external URL links
- Removing unhelpful characters, e.g., non-alphabetical characters.
- Converting capital letter to lowercase

Preprocessing

- **Stemming:** wordform stripped of some characters
- **Lemmatization:** the base (or citation) form of a word
- Example:
 - ordered could fill pandora bracelet right away wan na wait holiday filled liked fact lot pink charm barely got shipment still charm bracelet wear often since turn wrist green

```
ps = nltk.stem.porter.PorterStemmer()
print([ps.stem(word) for word in txt])
```

```
['order', 'could', 'fill', 'pandora', 'bracelet', 'right', 'away', 'wan', 'na', 'wait', 'holi  
day', 'fill', 'like', 'fact', 'lot', 'pink', 'charm', 'bare', 'got', 'shipment', 'still', 'ch  
arm', 'bracelet', 'wear', 'often', 'sinc', 'turn', 'wrist', 'green']
```

```
ps = nltk.stem.WordNetLemmatizer()
print([ps.lemmatize(word) for word in txt])
```

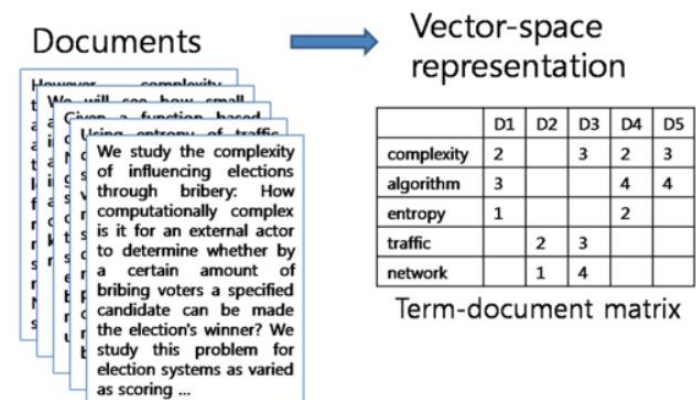
```
['ordered', 'could', 'fill', 'pandora', 'bracelet', 'right', 'away', 'wan', 'na', 'wait', 'ho  
liday', 'filled', 'liked', 'fact', 'lot', 'pink', 'charm', 'barely', 'got', 'shipment', 'stil  
l', 'charm', 'bracelet', 'wear', 'often', 'since', 'turn', 'wrist', 'green']
```

Feature Extraction

- We start from raw data, e.g., text, and build values, i.e., features, intended to be informative and non-redundant to help the subsequent learning and generalization steps.
 - Is a non-trivial task and specific to the application
 - The quality is judged by performance
 - A tedious task as applications become more abstract, e.g., document categorization vs dialogue system
- Example: Are you satisfied with our service?

Bag of Words (BoW)

- Simple example: counting the number of words in the input text
- Bag of words:
 - we pick a dictionary of words and then put an arbitrary order on the words, e.g., alphabetical order
 - We convert the text into a feature vector by reporting the frequency of occurrence of the words in the text
- Ex:
 - Dic = [good, bad, nice, expensive, love]
 - [I love this shirt because it is nice and worm. The color also is nice and matches my skin tone] -> [0, 0, 2, 0, 1]
 - Use case?



Bag of Words (BoW)

- Limitations of bag of word:
 - Insensitive to language structure: “I wanna eat ice cream” vs “wanna eat ice cream?”
 - Information in word dependencies is overlooked: “new york” vs “new book”
 - The resulting vectors are highly sparse which leads to high computational costs
- Why do we use BoW?

Term Frequency - Inverse Document Frequency

- Core idea: reflect how important a word is to an instance in the dataset.
- Term Frequency: a measure of how frequently a term, t , appears in an instance (document), d :

$$tf_{t,d} = \frac{n_{t,d}}{\text{Number of terms in the document}}$$

- The frequency of occurrence is normalized
- Inverse Document Frequency: a measurement of how important a term is in the dataset.

$$idf_t = \log \frac{\text{number of documents}}{\text{number of documents with term 't'}}$$

Term Frequency - Inverse Document Frequency

- TF-IDF score: reflects importance of the word for a particular instance and informative to categorize the documents

$$(tf_idf)_{t,d} = tf_{t,d} * idf_t$$

- We use TF-IDF to extract the feature vector
- Ex: What is TF-IDF score for stop words?

Implemented in NLTK

Statistical Learning

- Primary Goal: finding a **predictive** function based on an annotated training dataset using probability theory $(\mathbf{x}_i, \mathbf{y}_i) \sim p(\mathbf{x}, \mathbf{y})$
- Setup: given the value of an input vector X , i.e., features, make a good prediction \hat{Y} of the output Y (i.e., $\hat{Y} = Y$ with high probability) using the predictive function
- Maximum posterior estimation: $P(Y|X)$

Generative vs. Discriminative Models

Generative

- Learn a model of the joint probability $P(X, Y)$
- Use Bayes' Rule to calculate $P(Y|X)$
- Return the class that most likely to have generated that instance
- Examples: Naïve Bayes

Discriminative

- Model posterior probability $P(Y|X)$ directly
- Class is a function of feature vector
- Find the exact function that minimizes classification errors on the training dataset and use it for prediction
- Examples: Linear classifier, Logistic regression, Neural Networks (NNs), Support Vector Machines (SVMs)

Discriminative vs. Generative Classifiers

- Discriminative classifiers are generally more effective, since they directly optimize the classification accuracy. But
 - They are all sensitive to the choice of features, and in traditional ML these features are extracted heuristically
 - Overfitting can happen if data is sparse
- Generative classifiers directly model the joint probability which is helpful when generating text is necessary but:
 - Modeling the joint probability is a harder problem than classification if only classification is our goal
 - They can easily exploit unlabeled data

Bayes Classifier

- Bayes Rule:

$$\begin{array}{ccc} \text{Posterior} & & \text{Prior} \quad \text{Likelihood} \\ P(Y|X) & = & \frac{P(Y)P(X|Y)}{P(X)} \end{array}$$

- Bayes Optimal Classifier:

$$\hat{Y} = \arg \max_Y P(Y)P(X|Y)$$

- We use multiple features

$$\hat{Y} = \arg \max_Y P(Y)P(X_1, \dots, X_n|Y)$$

(Multinomial) Naïve Bayes Classifier

- Challenges for the Bayes optimal classifier
 - Computing the joint likelihood probability is practically almost impossible
- Assuming statistical independence between the features:

$$\begin{aligned}\hat{Y} &= \arg \max_Y P(Y)P(X_1, \dots, X_n|Y) \\ &= \hat{Y} = \arg \max_Y P(Y)\prod_{i=1}^n P(X_i|Y)\end{aligned}$$

(Multinomial) Naïve Bayes Classifier

- Challenges for the Naïve Bayes classifier
 - Computing the likelihood probability even for single features is practically difficult because many words are not frequently used: superb vs good

$$\hat{Y} = \arg \max_Y P(Y)P(X_1, \dots, X_n|Y)$$

- Zero probabilities cannot be conditioned away, no matter the other helpful evidence!
- Smoothing: we add small non-zero probabilities to avoid zero probabilities

Example

- Features = {I , hate, love, this, shirt}

$$\sim \{X1, X2, X3, X4, X5\}$$

- Training

– I hate this shirt (Y=0)

– Love this shirt (Y=1)

$$\text{Features} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

- What is $P(Y|X)$?

$$- P(X_i=k|Y=j) = \#(\text{documents with } X_i=k \text{ and } Y=j) / \#(\text{documents with } Y=j)$$

$$P(X|Y) = \begin{bmatrix} P(X1=0|Y=0) & P(X2=0|Y=0) & P(X3=0|Y=0) & P(X4=0|Y=0) & P(X5=0|Y=0) \\ P(X1=0|Y=1) & P(X2=0|Y=1) & P(X3=0|Y=1) & P(X4=0|Y=1) & P(X5=0|Y=1) \\ P(X1=1|Y=0) & P(X2=1|Y=0) & P(X3=1|Y=0) & P(X4=1|Y=0) & P(X5=1|Y=0) \\ P(X1=1|Y=1) & P(X2=1|Y=1) & P(X3=1|Y=1) & P(X4=1|Y=1) & P(X5=1|Y=1) \end{bmatrix}$$

- Prior $p(Y)$ $P(Y) = [P(Y=0) \quad P(Y=1)]$

$$- P(Y=j) = \#(\text{documents with } Y=j) / \#(\text{all documents})$$

- Testing

– hate shirt {x2,X5}

$$P(Y|X) \propto [P(Y=0) \times P(X2=1|Y=0) \times P(X5=1|Y=0) \quad P(Y=1) \times P(X2=1|Y=1) \times P(X5=1|Y=1)]$$

Example

- Features = {I, hate, love, this, shirt}
- Training

– I hate this shirt

– Love this shirt

- What is $P(Y|X)$?

- Prior $p(Y)$

- Testing

– hate shirt

$$P(X|Y) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

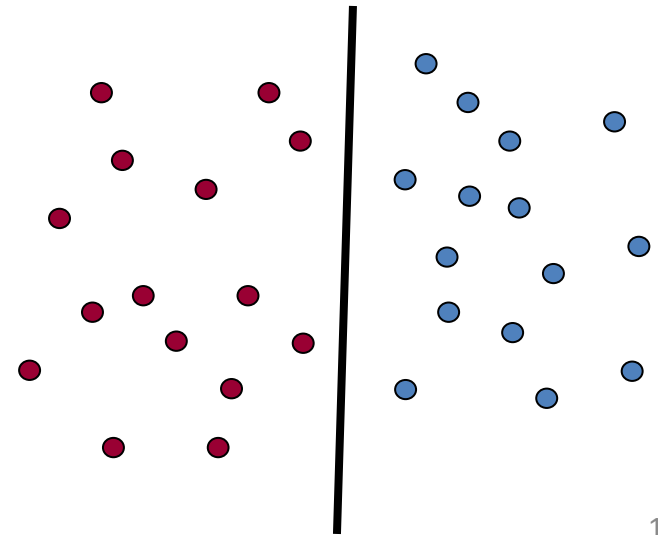
$$P(Y) = [1/2 \quad 1/2]$$

$$P(Y|X) \propto [1/2 \times 1 \times 1 \quad 1/2 \times 0 \times 1] \propto [1 \ 0]$$

Implemented in sklearn

Linear Classifier

- An interpretation for discriminative models is that we find a boundary between the two classes in the **geometrical** features space
- A linear classifier assumes that the data points are linearly separable in the feature space
- The goal is to find a boundary



Linear models

- A linear function in n -dimensional space (i.e. we have n features) is define by $n+1$ weights:

$$Y = \sum_{i=0}^n \beta_i X_i$$

- We find the model weights such that the linear function acts as a good predictive model

- Is not necessarily unique!

