Natural Language Processing: CIS 410/510

Text Classification

Instructor: Thien Huu Nguyen

Based on slides from: Ralph Grishman, David Bamman, Fabrizio Sebastiani, Dan Jurasky, Chris Manning and others

Text Examples

"… is a film which still causes real, not figurative, chills to run along my spine, and it is certainly the bravest and most ambitious fruit of Coppola's genius"

Roger Ebert, Apocalypse Now

Is this review positive or negative?

Text Examples

"I hated this movie. Hated hated hated hated hated this movie. Hated it. Hated every simpering stupid vacant audience-insulting moment of it. Hated the sensibility that thought anyone would like it."

Roger Ebert, North

What's about this?

Sentiment Analysis

- Given a text (e.g., a customer review about some product), we want to determine whether it is positive or negative (or both/neither)?
- This is an example of text classification where we want to classify a given text according to some predefined set of classes/types/labels (i.e., positive and negative in our example)

Different Types of Text Classification

- Depending on the nature of the predefined label sets:
  - Topics (e.g., politics, sports, science): by far the most frequent case, its applications are ubiquitous
  - Sentiment (e.g., positive, negative, neutral): useful in market research, online reputation management, customer relationship management, social sciences, political sciences
  - Languages (i.e., language identification): useful in query processing with search engines
  - Authors (i.e., authorship attribution): useful in forensics and cybersecurity

Applications of Text Classification

- Knowledge Organization; e.g.,
  - Classifying news articles for selective dissemination
  - Classifying scientific papers into specialized taxonomies
  - Classifying patents
  - Classifying answers to open-ended questions
  - Classifying topic-related tweets by sentiment

Applications of Text Classification

- Filtering: detecting some type of text for further investigation (treated as a classification between NonRelevant and Relevant types)
  - Spam Filtering: distinguish between legitimate and spam emails/messages
  - Detecting unsuitable content (e.g., porn, violent content, racist content, fake news): an important application recently (e.g., to interfere social media)

The rule-based approach for text classification

Training Examples	Labels
Simply loved it	Positive
Most disgusting food I have ever had	Negative
Stay away, very disgusting food!	Negative
Menu is absolutely perfect, loved it!	Positive
A really good value for money	Positive
This is a very good restaurant	Positive
Terrible experience!	Negative
This place has best food	Positive
This place has most pathetic serving food!	Negative

- Mostly if-else rules based on linguistic intuition and corpus examination (e.g., *if text involves "disgusting", "terrible" then the label is negative*)
- Although many extensions are possible
- Disadvantages: expensive to setup and maintain, hard to switch to different domains/labels/languages

Machine learning for text classification

- Supersede the rule-based approach
  - A generic (task-independent) learning algorithm is used to train a classifier from a set of manually classified examples
  - The classifier learns, from these training examples, the characteristics a new text should have in order to be assigned to some label
- Advantages
  - Annotating/locating training examples is in general cheaper than writing classification rules
  - Easy updates to changing conditions (e.g., changing the label set, domains etc.)

Machine learning for text classification

Training Examples	Labels
Simply loved it	Positive
Most disgusting food I have ever had	Negative
Stay away, very disgusting food!	Negative
Menu is absolutely perfect, loved it!	Positive
A really good value for money	Positive
This is a very good restaurant	Positive
Terrible experience!	Negative
This place has best food	Positive
This place has most pathetic serving food!	Negative

A sequence of words/characters (assuming tokenization)

$X = \{w_1, \ldots, w_n\}$

The label set

$c \in C = \{t_1, \ldots, t_K\}$

- Training dataset $D = \{(X_1, c_1), (X_2, c_2), \ldots, (X_N, c_N)\}$ (pairs of input texts and the corresponding labels)
- There are also disjoint development dataset (to choose the best hyper-parameters for the machine learning models) and test dataset (used only once to evaluate and compare the performance of the final models)

Machine learning for text classification

- From the training dataset $D = \{(X_1, c_1), (X_2, c_2), \dots, (X_N, c_N)\}$, we want to learn a model/classifier/function that can predict the label for a new input text X (the classification problem):

$$f: X \rightarrow c \ \in C$$

- In machine learning, this is often done by computing the probability distribution over the possible class in C given the input document X:

$$P(c|X)$$

- The label for o new document X can then determined via the $argmax$ function:

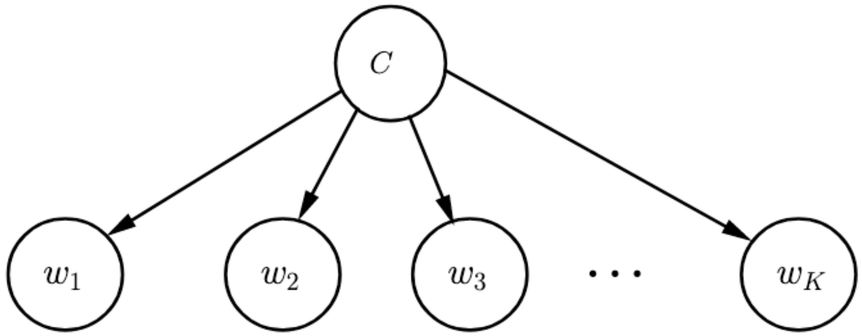
$$c_* = argmax_c \, P(c|X)$$

The Naïve Bayes Classifier

- Using Bayes's Rule

$$argmax_c P(c|X) = argmax_c \frac{P(X|c)P(c)}{P(X)}$$
$$= argmax_c P(X|c)P(c)$$
$$= argmax_c P(w_1, \dots, w_n|c)P(c)$$
$$= argmax_c P(w_1|c) \dots P(w_n|c)P(c)$$

- The last step is based on:
  - bag-of-word representation: the positions of the words do not matter; a document is represented by the set of words it contains.
  - naïve assumption of independence of the word probabilities given the class.



The Naïve Bayes Classifier

- We then estimate these probabilities from the training data D using maximum likelihood estimators:

- The Bernoulli model: only use the presence/absence of a word/term in a document as feature:

$$P(c) = \frac{\text{count}(docs\ labeled\ c\ in\ D)}{\text{count}(docs\ in\ D)}$$

: probability that a document is labeled c

$$P(w_i|c) = \frac{\text{count}(docs\ labeled\ c\ containing\ w_i\ in\ D)}{\text{count}(docs\ labeled\ with\ c\ in\ D)}$$

: probability that a document labeled c contains w_i

The Naïve Bayes Classifier

- The multinomial model: based on the frequency of terms in the documents:

$$P(w_i|c) = \frac{\text{count}(instances\ of\ w_i\ in\ docs\ labeled\ c\ in\ D)}{total\ length\ of\ docs\ labeled\ c\ in\ D}:$$

probability that a word in a doc labeled c is w_i

- Often has better performance on long documents

A problem

- Consider the sentiment analysis problem with only two classes "*positive*" and "*negative*"

- Suppose a glowing review *GR* (with lots of positive words) includes one word, "*mathematical*", previously seen only in negative reviews

- What is $P(positive|GR)$?

A problem

- $P(positive|GR) = 0$ as $P(\text{“}mathematical\text{”}|positive) = 0$
- The maximum likelihood estimate is poor when there is very little data
- We need to ‘smooth’ the probabilities to avoid this problem
  - By adding 1 to each count (Laplace (add-1) smoothing)

The Naïve Bayes Classifier

- The multinomial model: based on the frequency of terms in the documents with add-1 smoothing:

$$P(w_i|c) = \frac{1 + \mathrm{count}(instances\ of\ w_i\ in\ docs\ labeled\ c\ in\ D)}{d + total\ length\ of\ docs\ labeled\ c\ in\ D}:$$

probability that a word in a doc labeled c is w_i

where d is the number of words in the vocabulary extracted from training data.

Multinomial Naïve Bayes: Learning

- From training corpus, extract *Vocabulary*

- Calculate $P(c_j)$ terms
  - For each c_j in C do

    $docs_j \leftarrow$ all docs with class $=c_j$

$$P(c_j) \leftarrow \frac{|docs_j|}{|\text{total \# documents}|}$$

- Calculate $P(w_k \mid c_j)$ terms
  - $Text_j \leftarrow$ single doc containing all $docs_j$
  - For each word w_k in *Vocabulary*

    $n_k \leftarrow$ # of occurrences of w_k in $Text_j$

$$P(w_k \mid c_j) \leftarrow \frac{n_k + \alpha}{n + \alpha |Vocabulary|}$$

Example

$$P(c) = \frac{N_c}{N}$$

$$P(w \mid c) = \frac{count(w,c)+1}{count(c)+|V|}$$

	Doc	Words	Class
Training	1	Chinese Beijing Chinese	c
	2	Chinese Chinese Shanghai	c
	3	Chinese Macao	c
	4	Tokyo Japan Chinese	j
Test	5	Chinese Chinese Chinese Tokyo Japan	?

Priors:

$P(c) = \frac{3}{4}$

$P(j) = \frac{1}{4}$

Conditional Probabilities:

P(Chinese|*c*) = (5+1) / (8+6) = 6/14 = 3/7

P(Tokyo|*c*) = (0+1) / (8+6) = 1/14

P(Japan|*c*) = (0+1) / (8+6) = 1/14

P(Chinese|*j*) = (1+1) / (3+6) = 2/9

P(Tokyo|*j*) = (1+1) / (3+6) = 2/9

P(Japan|*j*) = (1+1) / (3+6) = 2/9

Choosing a class:

$P(c|d5) \propto 3/4 * (3/7)^3 * 1/14 * 1/14 \approx 0.0003$

$P(j|d5) \propto 1/4 * (2/9)^3 * 2/9 * 2/9 \approx 0.0001$

https://text-processing.com/demo/

Problems with Naïve Bayes Classifier: Ambiguous terms

- A word can be interpreted as "positive" or "negative" depending on context. For example:
  - "*low*" can be positive: "*low price*"
  - "*low*" can also be negative: "low quality"

- Modeling words independently and ignoring their order might not be able to capture such context dependence.

Problems with Naïve Bayes Classifier: Negation

- How can we handle:
  - *"the equipment never failed"*
- If *"failed"* is not attached to *"never"*, it will create a very different sense of sentiment.
- A simple trick:
  - Modify words following negation:
    - *"the equipment never failed"* → *"the equipment never NOT_failed"*
  - Treat them as a separate "negated" vocabulary

Negation: How far to go?

"the equipment never failed and was cheap to run"

"the equipment never NOT_failed NOT_and NOT_was NOT_cheap NOT_to NOT_run"

- Have to determine the scope of negation!

Summary: Naïve Bayes is not so naïve

- Very fast, low storage requirements
- Robust to irrelevant features
  - Irrelevant features cancel each other without affecting results
- Very good in domains with many equally important features
- Optimal if the independence assumptions hold:
  - If assumed independence is correct, then it is the Bayes Optimal Classifier for the problem
- A good dependable baseline for text classification
  - But we will see other classifiers that give better accuracy with better handling of:
    - Ambiguous terms
    - Negation
    - Comparative reviews
    - Revealing aspects of an opinion:
      - *the car looked great and handled well, but the wheels kept falling off*

Divergence: Information Retrieval

- Task: given query = list of keywords, identify and rank relevant documents from collection
- Basic idea: find documents whose set of words most closely matches words in query

Topic Vector

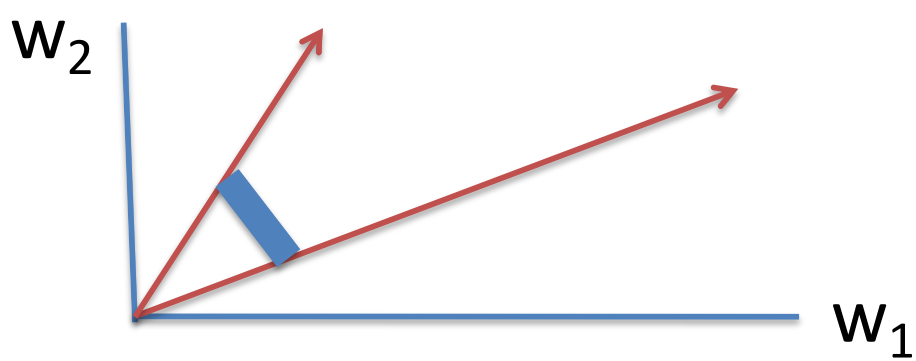
- Suppose the document collection has n distinct words (the vocabulary): $w_1, \dots, w_n$
- Each document is characterized by an n-dimensional vector whose i^{th} component is the frequency of word w_i in the document (i.e., term frequencies – *tf*)

Example

- X_1 = [The cat chased the mouse.]
- X_2 = [The dog chased the cat.]
- W = [The, chased, dog, cat, mouse] (n = 5)
- V_1 = [2 , 1 , 0 , 1 , 1]
- V_2 = [2 , 1 , 1 , 1 , 0]

- Given a query Q, compute its corresponding topic vector, and rank documents according to the cosine similarity between Q's vector and the documents' vectors.

$$sim(A,B) = \frac{\sum_i a_i \times b_i}{\sqrt{\sum_i a_i^2} \times \sqrt{\sum_i b_i^2}}$$



Weighting the components

- Unusual words like *"elephant"* determine the topic much more than common words such as "*the*" or "*have*".

- We can emphasize the important words by:
  - Ignore words on a stop list (e.g., "*the*", "*a*")
  - Weight each term frequency tf_i by its inverse document frequency (idf_i):

$$idf_i = \log(\frac{N}{n_i})$$

where N = the size of the collection and n_i = the number of documents containing the i^{th} term.

$$w_i = tf_i \times idf_i$$

Back to machine learning for text classification

- Naïve Bayes can be extended to include more features than just the words/terms in the text themselves, e.g.,
  - Words in the title
  - Author, length, date of document
  - Sender, recipient of email
  - Noun phrases or *n*-grams
  - Number of punctuation marks
  - …
- However, the more features we include, the more likely they have dependencies with each other, thus violating the independency assumption of Naïve Bayes.
  - We need methods that can handle the inter-dependency between features, thus allowing us to introduce as many features as we want to reflect our intuition about the problem.

Machine learning for text classification

- Given a document/text, using the features we designed, we convert it into a vector (called the representation vector)
- Each dimension corresponds to one feature.

X = [The dog chased the cat.]

Feature Engineering

Features	the	dog	chased	mouse	cat	length	appCap	authorIsTom	authorisThien
Values	2	1	1	0	1	6	0	0	1

- Normalization is helpful.
- Can involve conjunction features, e.g., n-grams, combination of a feature and a label, to emphasize the cooccurrence of the features (thus highly inter-dependent)

A more formal description for text classification

- In the first step, a document X is transformed into a vector $R(X) = [R_1(X), R_2(X), \dots, R_d(X)]$
- $R(X)$ caputres the important/representative features for the classification task (e.g., feature engineering)
- In traditional machine learning for NLP, $R_i(X)$ is often binary and manually designed by domain experts (thus being more interpretable)
- The distribution $P(c|X)$ is then computed via $R(X)$ following some parameterized functions (e.g., the linear function):
$$P(c|X) = S(R(X), \theta)$$
- Choosing the form of $S(X)$ is called designing the model (an important step)

A more formal description for text classification

- Given the function $S(R(X), \theta)$, the classification problem becomes an optimization problem to determine the suitable values for θ

- For NLP, the optimization problem for θ is:

$\theta^* = argmin_\theta E_{(X,c)\backsim P(X,c)}[L(S(R(X), \theta), c)]$ (i.e., minimal error on every possible pair of input and output)

where: $P(X, c)$ is the joint distribution for the input X and output/label y

$L(S(R(X), \theta), c)$ is the cost function that evaluates the loss of using $S(R(X), \theta)$ to determine the label for X (the predicted type) given that y is the correct label.

A more formal description for text classification

- The computation of $E_{(X,c)\backsim P(X,c)}[L(S(R(X),\theta),c)]$ requires the enumeration over all possible pairs of (X,c) (thus intractable)

- In practice, we obtain a training dataset $D = \{(X_1,c_1),(X_2,c_2),\dots,(X_n,c_n)\}$, leading to the empirical distribution for (X,c): $\hat{P}(X,c)$. Thus, θ^* can be computed by:

$$\begin{aligned}\theta^* &= argmin_\theta E_{(X,c)\backsim P(X,c)}[L(S(R(X),\theta),c)]\\ &\approx argmin_\theta E_{(X,c)\backsim \hat{P}(X,c)}[L(S(R(X),\theta),c)]\\ &= argmin_\theta \frac{1}{n}\sum_{i=1}^{n} L(S(R(X_i),\theta),c_i)\end{aligned}$$

Logistic regression -- maxent

- The parameter θ consists of two elements, i.e., the matrix $B \in \mathbb{R}^{d \times K}$ and the bias vector $b \in \mathbb{R}^{K}$ ($\theta = \{B, b\}$).

- The *i*-th column of B corresponds to the feature weights for the *i*-th label c_i of C.

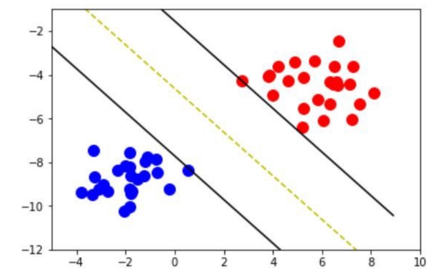
- Given the parameters, the likelihood vector A for the types of Y is computed via:
$$A = B^T R(X) + b = [a_1, a_2, \dots, a_K]$$

- Finally, the likelihood vector is normalized using the softmax function to obtain the probability distribution:
$$S(R(X), \theta) = softmax(A) = \left[\frac{e^{a_1}}{e^Z}, \frac{e^{a_2}}{e^Z}, \dots, \frac{e^{a_n}}{e^Z}\right], Z = \sum_{i=1}^{K} e^{a_i}$$

- The loss function in this case:
$$L(S(R(X), \theta), c) = -\log S(R(X), \theta)[c] = -\log\left[\frac{e^{a_c}}{e^Z}\right]$$

Support vector machines (SVM)

- For simplicity, assume the binary classification setting (only two types, denoted by -1 and 1, in C, i.e., $\mathrm{C} = \{-1,1\}$) (for $|C| > 2$, we can consider multiple binary classification problems)
- Each input X is seen as a point in the d-dimensional space defined by its vector $R(X)$
- For SVM, the goal is to find a hyperplane that divides the group of training instances X_i with $c_i = 1$ and the group of those with $c_i = -1$
- As there might be multiple satisfying hyperplanes, SVM seeks to find two parallel hyperplanes that separate the instances and have the largest distance between them
- The final hyperplane is then the one that stands in the middle of such two hyperplanes

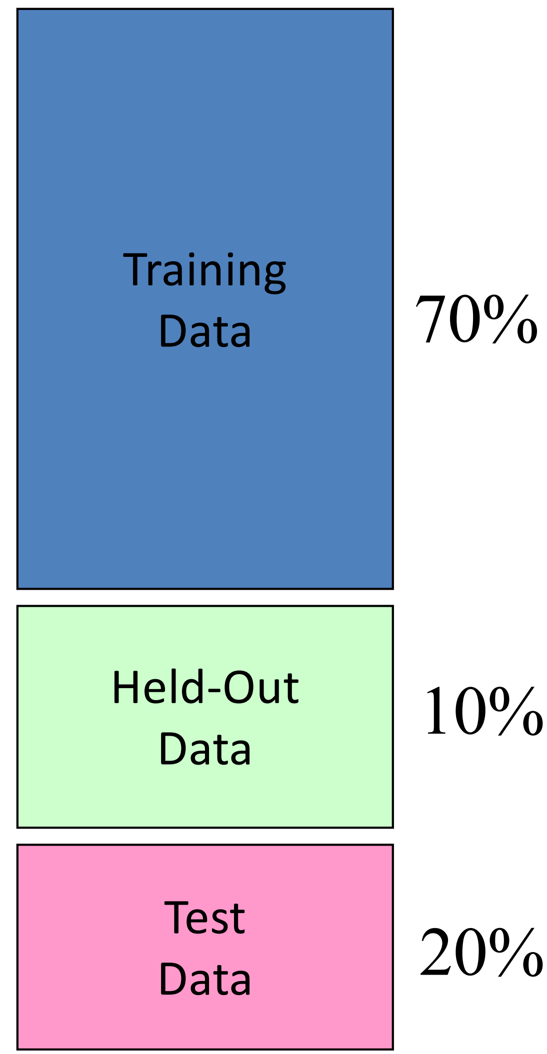


Support vector machines (SVM)

- This translates into the score function $A(R(X),\theta)$ that are parameterized by a weight vector B ($|B| = |R(X)|$) and a bias b ($\theta = [B, b]$):
$$A(R(X),\theta) = B^T R(X) - b$$
- The probability distribution is then simply:
$$S(R(X),\theta) = [0.5 - A(R(X),\theta), 0.5 + A(R(X),\theta)]$$
- The loss function in this case (the hinge loss):
$$L(S(R(X),\theta), y) = \max(0, 1 - yA(R(X),\theta))$$
- The description of SVM so far can only work for the problems where the two types of data can be separated by hyperplanes (i.e., linearly separable).
- For nonlinear separation, we need to incorporate the kernel trick (i.e., mapping the original spaces into another space where the data becomes linearly separable by building a kernel function).
    - Often done by defining a kernel function

How to evaluate the models?

- Data: labeled examples, e.g. emails marked spam/ham
  - Training set
  - Held out set/Development set (dev set)
  - Test set
  - These sets are disjoint!
  - Can also do cross-validation over multiple splits
    - Pool results over each split
    - Compute average dev set result
- Features: attribute-value pairs which characterize each *X*
- Experimentation cycle
  - Learn parameters (e.g. model probabilities) on training set
  - (Tune hyperparameters on held-out set)
  - Compute accuracy of test set
  - Very important: never "peek" at the test set!
- Evaluation
  - Accuracy: fraction of instances predicted correctly
- Overfitting and generalization
  - Want a classifier which does well on *test* data
  - Overfitting: fitting the training data very closely, but not generalizing well (i.e., poor results on the test set)
  - Common ways to improve generalization:
    - L1, L2 regularization
    - Cross-validation
    - Feature Selection
    - Including more data!



Precision, Recall, F1

- Suppose that we are doing spam detection for emails and there are only two classes in our classification problem (i.e., spam or not spam).
- We can evaluate the models' performance by accuracy (the fraction of emails in the test set that are correctly predicted).
- But we don't really care about the ham emails. We want evaluation measures that focus directly on the spam emails. So, we use the confusion matrix:

- Accuracy = (TN + TP) / total = (50+100)/165 = .91
- Precision (P) = % predicted examples that are correct = TP / (TP + FP) = 100 / (100 + 10) = .91
- Recall (R) = % of correct examples that are selected = TP / (TP + FN) = 100 / (100 + 5) = .95
- F1 = 2PR/(P+R) – a trade-off between precision and recall

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Evaluation with more than two classes

- Confusion matrix: for each pair of classes <c_1, c_2>, how many documents from c_1 were incorrectly assigned to c_2?

Docs in test set	Assigned UK	Assigned poultry	Assigned wheat	Assigned coffee	Assigned interest	Assigned trade
True UK	95	1	13	0	1	0
True poultry	0	1	0	0	0	0
True wheat	10	90	0	1	0	0
True coffee	0	0	0	34	3	7
True interest	-	1	2	13	26	5
True trade	0	0	2	14	5	10

- Macroaveraging: compute performance for each class, then average (classes are equal)
- Microaveraging: collect decisions for all classes, compute confusion table, evaluate (more preferable if classes are imbalanced)

Recall:

Fraction of docs in class *i* classified correctly: $\frac{c_{ii}}{\sum_j c_{ij}}$

Precision:

Fraction of docs assigned class *i* that are actually about class *i*: $\frac{c_{ii}}{\sum_j c_{ji}}$

Accuracy: (1 - error rate)

Fraction of docs classified correctly: $\frac{\sum_i c_{ii}}{\sum_j \sum_i c_{ij}}$

Micro- vs. Macro-Averaging: Example

Class 1

	Truth: yes	Truth: no
Classifier: yes	10	10
Classifier: no	10	970

Class 2

	Truth: yes	Truth: no
Classifier: yes	90	10
Classifier: no	10	890

Micro Ave. Table

	Truth: yes	Truth: no
Classifier: yes	100	20
Classifier: no	20	1860

- Macroaveraged precision: (0.5 + 0.9)/2 = 0.7
- Microaveraged precision: 100/120 = .83
- Microaveraged score is dominated by score on common classes

Some datasets for text classification

- Reuters-21578 (http://disi.unitn.it/moschitti/corpora.htm)
- 20Newsgroups (http://disi.unitn.it/moschitti/corpora.htm)
- Yelp reviews 2013, 2014, 2015 (http://ir.hit.edu.cn/~dytang/paper/emnlp2015/emnlp-2015-data.7z)
- .....

Recipe for the real world

- No training data
  - Use manually written rules (although time-consuming and human need to tune on the dev set)
- Very little data
  - Use Naïve Bayses (a high-bias algorithm)
  - Try to get more labeled data with some clever way
  - Use semi-supervised learning (e.g., bootstrapping)
- A reasonable amount of data
  - SVM, logistic regression, deep learning, …
- A huge amount of data
  - SVM, logistic regression, deep learning, …
  - With enough data, classifier may not matter