Text classification

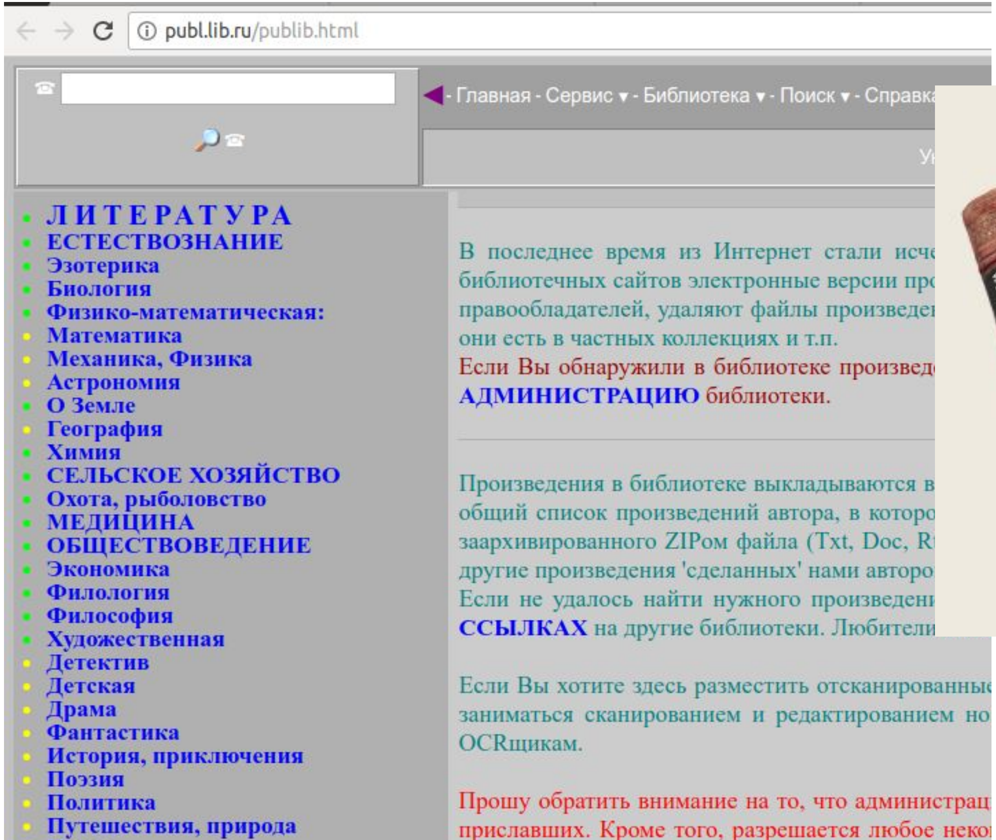
Anton Alekseev
Steklov Mathematical Institute, St Petersburg Department

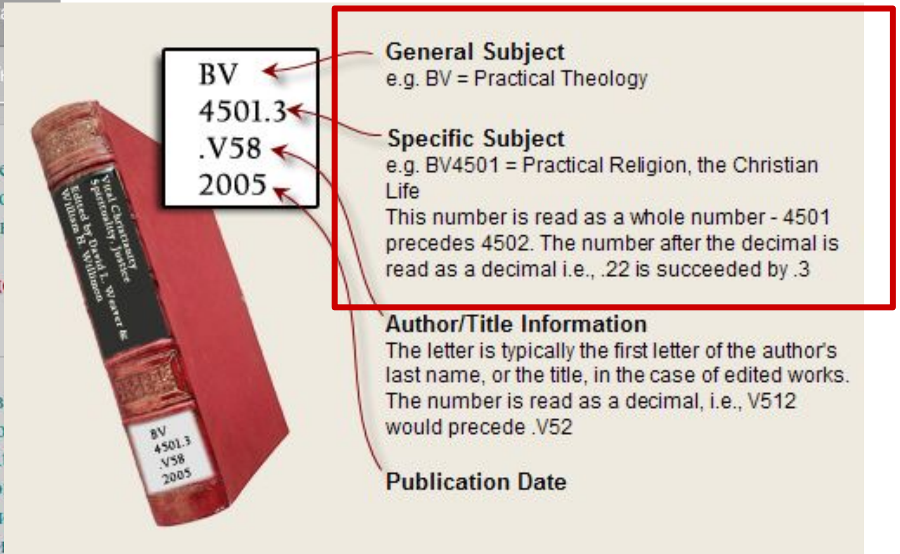
NRU ITMO, St Petersburg, 2019
anton.m.alexeyev+itmo@gmail.com

Plan

1. Motivation
2. Classification task
3. Classifier example: Naive Bayes
4. Classification quality evaluation
5. Classification methods review
    a. Linear methods
    b. Metric methods
    c. Logical methods
    d. Ensembles
6. Typical tasks and special cases

Motivation: roots





http://kencozmo.blogspot.ru/2011/09/chapter-3.html

Motivation

1. **Sentiment analysis**: track/check if the users are happy with the product or not
(optional: + find out which particular feature user [dis]liked)

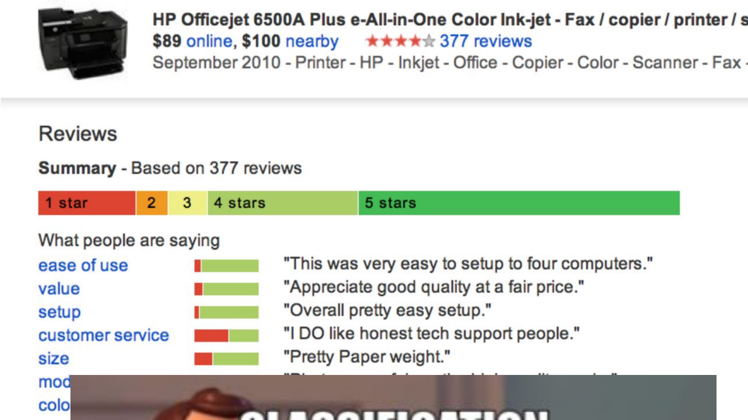
2. **Topic classification**: section the news article to be put in

3. **Spam detection**: predict if letters are unwanted by user based on those tagged by him/her as spam

4. **Incomplete data imputation**: predict user's gender based on text he/she publishes/likes/skips

5. **Many more**: authorship attribution, sociodemographic characteristics, etc...

Google Product Search



Plan

1. ~~Motivation~~
2. Classification task
3. Classifier example: Naive Bayes
4. Classification quality evaluation
5. Classification methods review
    a. Linear methods
    b. Metric methods
    c. Logical methods
    d. Ensembles
6. Typical tasks and special cases

Classification task

Supervised learning task. Given:

- a set of documents (texts) $\mathbf{D = \{ d_1, d_2, \dots d_n\}}$
- a set of classes (categories) $\mathbf{C = \{ c_1, c_2, \dots c_k \}}$
- usually there is a training set -- a subset of **D x C**, that is, document-class pairs

Task:

- train a function **f: D => C**, matching each document with the correct class

Plan

1. ~~Motivation~~
2. ~~Classification task~~
3. Classifier example: Naive Bayes
4. Classification quality evaluation
5. Classification methods review
    a. Linear methods
    b. Metric methods
    c. Logical methods
    d. Ensembles
6. Typical tasks and special cases

Classifier example: Naive Bayes

aka simple Bayes, independent Bayes

is called so thanks to being a straightforward Bayes theorem application

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Approach: learn **P(c|d)** and choose **c** with the largest conditional probability value for every **d**

class

$$p(C_k \mid x_1, \ldots, x_n)$$

words in a document

Assumption: all words are conditionally independent

$$p(x_i \mid x_{i+1}, \ldots, x_n, C_k) = p(x_i \mid C_k)$$

Naive Bayes: formulae

Class probability for the set of given words in a document

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

p(x) — constant!

...we care about the numerator only

$$p(C_k, x_1, \ldots, x_n)$$

Let's rewrite it using the chain rule

$$\begin{aligned}
p(C_k, x_1, \ldots, x_n) &= p(x_1, \ldots, x_n, C_k) \\
&= p(x_1 \mid x_2, \ldots, x_n, C_k) p(x_2, \ldots, x_n, C_k) \\
&= p(x_1 \mid x_2, \ldots, x_n, C_k) p(x_2 \mid x_3, \ldots, x_n, C_k) p(x_3, \ldots, x_n, C_k) \\
&= \ldots \\
&= p(x_1 \mid x_2, \ldots, x_n, C_k) p(x_2 \mid x_3, \ldots, x_n, C_k) \ldots p(x_{n-1} \mid x_n, C_k) p(x_n \mid C_k) p(C_k)
\end{aligned}$$

Naive Bayes: formulae

Using the conditional independence assumption, we get

$$\begin{aligned} p(C_k \mid x_1, \dots, x_n) &\propto p(C_k, x_1, \dots, x_n) \\ &\propto p(C_k)\, p(x_1 \mid C_k)\, p(x_2 \mid C_k)\, p(x_3 \mid C_k)\, \cdots \\ &\propto p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,. \end{aligned}$$

classifier is ready:

$$\hat{y} = \underset{k \in \{1, \dots, K\}}{\operatorname{argmax}}\ p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k).$$

the so-called MAP (maximum a posteriori) decision rule

Naive Bayes: how to compute this

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \; p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

share of words labeled with C_k in the training set

fraction of the word x_i among all texts labeled with C_k

1. Estimate probabilities using the formula above
2. Compute the value for every class for every new incoming document
3. Choose the class with the largest value

Yes, doing **smoothing** does make sense here; one can also consider taking

- a **share of documents containing the word** (binary Naive Bayes) instead of plain word frequencies
- log-frequencies instead of plain word frequencies

Naive Bayes: discussion

- independence assumption
  (natural language is not a bag of words)
- weights of long documents differ a great deal

+ robust to unknown words
+ simple and fast
+ is often used as a simple baseline

Plan

1. ~~Motivation~~
2. ~~Classification task~~
3. ~~Classifier example: Naive Bayes~~
4. Classification quality evaluation
5. Classification methods review
    a. Linear methods
    b. Metric methods
    c. Logical methods
    d. Ensembles
6. Typical tasks and special cases

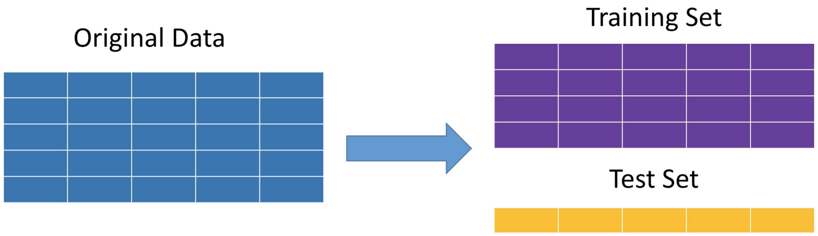
Machine learning models quality evaluation

We have the data, we have the metric

Splitting into

- train set
- test set

Believing these subsets are ‘sampled from the same distribution’ (otherwise training makes almost no sense)



https://jessesw.com/images/Rec_images/Traintest_ex.png

Machine learning models quality evaluation

Deadly Sin №1
Test data leaks into train set
(this way we lose generalization capability and estimates validity)

Deadly Sin №2
Tuning hyperparameters on test set

But how do we tune the parameters? Ideas?

DataFest sticker

Machine learning models quality evaluation

TRAIN	**DEV**	**TEST**

1. TRAIN - training model
2. DEV - evaluating quality + analyzing errors + tuning hyperparameters
3. TEST - blind quality evaluation: looking at quality metric ONLY + not too often, so as not to overfit

Binary classification quality evaluation

Example: **spam** (positive) or **not spam** (negative) emails

true labels

predictions		spam!	not spam!
	spam!	**TRUE POSITIVE** we’re happy	**FALSE POSITIVE** normal letter falling into a Spam folder **a tragedy**
	not spam!	**FALSE NEGATIVE** spam in inbox **not good**	**TRUE NEGATIVE** we’re happy

Binary classification quality evaluation

	gold positive	gold negative	
system positive	**true positive**	**false positive**	**precision** $= \frac{\text{tp}}{\text{tp+fp}}$
system negative	**false negative**	**true negative**	
	recall $= \frac{\text{tp}}{\text{tp+fn}}$		**accuracy** $= \frac{\text{tp+tn}}{\text{tp+fp+tn+fn}}$

F-measure

$$F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

$$F_1 = \frac{2PR}{P+R}$$

F1-measure - harmonic mean of recall and precision

Choose the target class and consider its prediction
a **positive** case;
Correct prediction — **true positive,** incorrect — **false positive**
+ the same for the other class

Binary classification quality evaluation

Let’s say **1** is a target class

ground_truth	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**
prediction	1	0	0	1	0	1	1	1

TP = 3
FP = 2
TN = 2
FN = 1

Accuracy = (3 + 2) / (3 + 2 + 2 + 1) = 0.625
Precision = 3 / (3 + 2) = 0.6
Recall = 3 / (3 + 1) = 0.75
F1 = 2 * 0.6 * 0.75 / (0.6 + 0.75) = 0.66(6)

Binary classification quality evaluation

accuracy = share of correct hits, is in [0, 1]

- won't tell us much if samples counts of different classes shares are imbalanced

precision = a share of truly **positive** among predicted as positive ones
recall = a share of truly positive that were actually predicted as positive ones

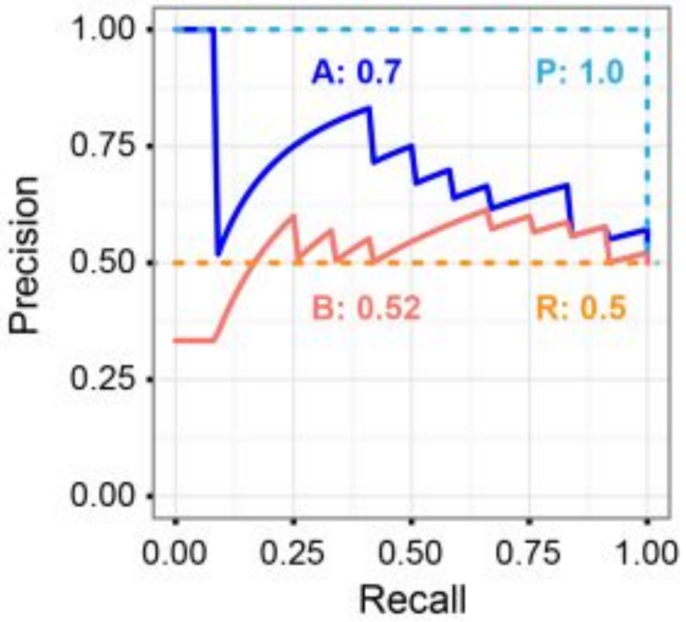
Checking your understanding: if the classifier sets all labels as the target class (all samples are predicted as positive ones), what are **precision and recall**?

Binary classification quality evaluation

Precision-Recall Curve

we change the parameter that changes precision and recall and look at the behaviour of precision and recall values

(this parameter is usually a probability threshold in a decision rule)



https://classeval.wordpress.com/introduction/introduction-to-the-precision-recall-plot/

Classification quality evaluation: multi-class

= number of classes > 2

1. **Accuracy**
   share of correctly predicted cases

2. **Micro-averaging**: Precision, Recall, FScore
   first we compute TP, FP, …, for every class
   and then we compute metrics values, summing all TPs, FPs, etc.

3. **Macro-averaging aka “all classes are equally important”**: Precision, Recall, FScore
   computing Precision, Recall,... for every class,
   then averaging (summing and dividing by the number of classes)

Classification quality evaluation: multi-class

ground_truth	**1**	**2**	**0**	**2**	**0**	**1**	**0**	**1**
prediction	0	2	0	1	2	1	1	2

Label 0

TP = 1, FP = 1
FN = 2, TN = 4

Precision = 0.5
Recall = 0.33

Label 1

TP = 1, FP = 2
FN = 2, TN = 3

Precision = 0.33
Recall = 0.33

Label 2

TP = 1, FP = 2
FN = 1, TN = 4

Precision = 0.33
Recall = 0.5

Macro-averaging

Pr = (0.5+0.33+0.33) / 3 = 0.387
R = (0.5+0.33+0.33) / 3 = 0.387
F1 = 2PrR / (Pr + R) = 0.387

Micro-averaging

Pr = (1+1+1) / (1+1+1 + 1+2+2) = 0.375
R = (1+1+1) / (1+1+1 + 1+2+2) = 0.375
F1 = 2PrR / (Pr + R) = 0.375

Plan

1. ~~Motivation~~
2. ~~Classification task~~
3. ~~Classifier example: Naive Bayes~~
4. ~~Classification quality evaluation~~
5. Classification methods review
    a. Linear methods
    b. Metric methods
    c. Logical methods
    d. Ensembles
6. Typical tasks and special cases

Notes on standard text representation approaches

Method #1, Bag-of-words: one hot
~ one-hot-encoding / dummy coding: many interpretable features
“Hush now, baby, baby, don't you cry”

Bag-of-words: word counts (sklearn: CountVectorizer)
counts or relative frequencies instead of one-hot values

hush	now	baby	wall	do	not	you	oh	cry
1	1	2	0	1	1	1	0	1

Bag-of-words: weird numbers (sklearn: TfIdfVectorizer)
TF-IDF or other estimates of terms importance

Notes on standard text representation approaches

By ‘forgetting’ about word order we lose information, however, there is a simple way to at least try to take word order into account!

Bag-of-ngrams (sklearn vectorizers support this out-of-the-box, btw)
ngram = n terms in a row as a single term

“New York”
“New Deli”
“**not** cool”
“catch up with”

+ other reasons why word order has to be dealt with

BOW: specifics and takeaways

https://twitter.com/stanfordnlp/status/399551909595344896

tens/hundreds of thousands of sparse features; curse of dimensionality may be a problem:

1. have to filter terms and introduce penalties for the most frequent and rare ones;
   implemented in almost any toolbox, e.g. in sklearn;
   (including **stopwords** filtering: “useless/common words”)

2. should choose models working with large number of sparse features
   ~~one can’t simply solve all problems with *Random Forest!*~~

3. should always experiment with choosing N in Ngrams and weights for terms (one-hot/tfidf etc.)

BOW: specifics and takeaways

tens/hundreds of thousands of sparse features; curse of dimensionality may be a problem:

1. have to filter terms and introduce penalties for the most frequent and rare ones;
   implemented in almost any toolbox, e.g. in sklearn;
   (including **stopwords** filtering: “useless/common words”)

2. should choose models working with large number of sparse features
   ~~one can’t simply solve all problems with *Random Forest!*~~

3. should always experiment with choosing N in Ngrams and weights for terms (one-hot/tfidf etc.)

https://twitter.com/stanfordnlp/status/399551909595344896

When BoW may not be enough?

- Small data
  - Zipf’s law
  - Rich morphology =>
    not too many training samples
  - ...what if we lemmatize? =>
    sometimes we can’t neglect morphology

- Short texts
  - same reasons
  - + intuitively: the larger the text the more good word predictors it has

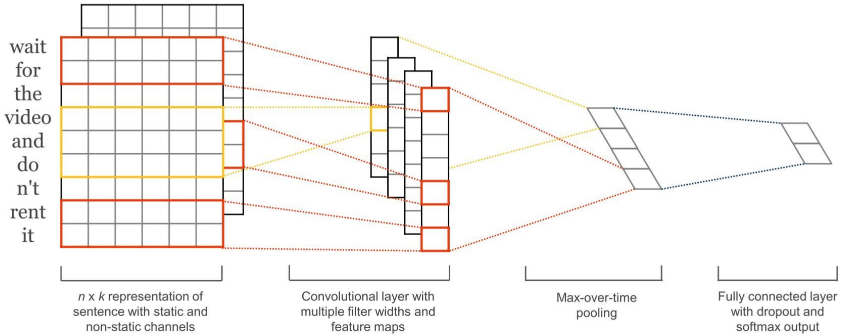
Trash bag // *Wikipedia*

https://upload.wikimedia.org/wikipedia/commons/thumb/c/c8/Garbage_bag.jpg/1200px-Garbage_bag.jpg

Notes on standard text representation approaches

Method #2 sum word vectors (e.g., word2vec) of all words in the texts
with weights proportional to importance weights (e.g. TF-IDF)

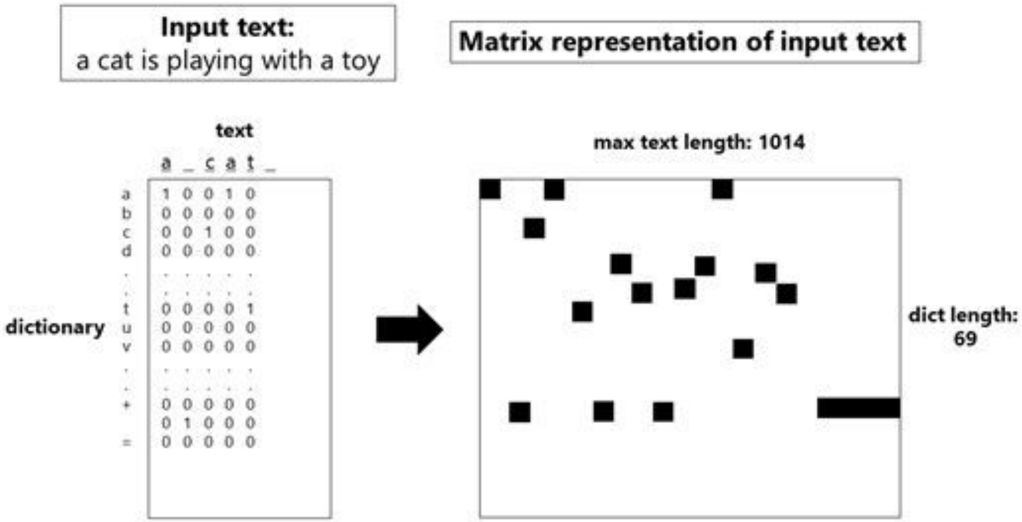
Method #3 concat word vectors (e.g., word2vec) of all words in the texts
into a matrix



What if we go beyond word level?

...that is, represent the text as a sequence of encoded characters (**Method #4**)

e.g. see: http://karpathy.github.io/2015/05/21/rnn-effectiveness/



https://blogs.technet.microsoft.com/machinelearning/2017/02/13/cloud-scale-text-classification-with-convolutional-neural-networks-on-microsoft-azure/

https://i.pinimg.com/originals/20/39/17/203917d3b4cd0fa531801d46a432d272.jpg

Representing texts

Custom features may also help:
POS counts, text length, weighted average word embeddings, RNN-based embeddings, etc.

Замечание о способах представления текстов

Способ #1, Bag-of-words: one hot (BOW; мешок слов)
~ one-hot-encoding / dummy coding: много интерпретируемых фич
"А не три, а не пять! Это надо знать!"

Bag-of-words: word counts (sklearn: CountVectorizer)
вместо единиц — частоты / относительные частоты

а	не	три	шесть	пять	это	надо	семь	знать
2	2	1	0	1	1	1	0	1

Bag-of-words: weird numbers (sklearn: TfIdfVectorizer)
вместо единиц — TF-IDF или другие оценки "значимости"

Замечание о способах представления текстов

Очевидно, с порядком слов в тексте мы теряем много информации, но есть простое средство для бедных!

Bag-of-ngrams (sklearn vectorizers поддерживают)
вместо отдельных термов наборы из n подряд идущих в тексте термов

"Нью Йорк"
"Нью Дели"
"**не** надо"
"catch up with"

BOW: специфика

много разреженных фич, можем столкнуться с проблемами больших размерностей, поэтому:

1. надо уметь фильтровать и наказывать термы весами; частотные, редкие и т. д.- это есть из коробки есть в sklearn; кроме того — фильтрация по словарям (в т. ч. **stopwords**: словарь "вредных слов")
2. выбираем модели для работы с большим числом разреженных признаков, ~~*не всё можно загонять в Random Forest!*~~
3. обязательно экспериментируем с числом N в ngram-max и вариациями one-hot/count/tf-idf/...

https://twitter.com/stanfordnlp/status/399551909595344896

30

Когда BoW плохо справляется?

- Мало обучающих данных
  - Закон Ципфа
  - Богатая морфология => слишком мало прецедентов для обучения
  - ...А если нормализуем => иногда теряем важную информацию
- Короткие тексты
  - Те же причины
  - + интуитивно: в большем тексте больше хороших слов-предикторов целевой переменной (or whatever)

Мешок для мусора // *Википедия*

https://upload.wikimedia.org/wikipedia/commons/thumb/c/c8/Garbage_bag.jpg/1200px-Garbage_bag.jpg

32

Conceptual stuff

In Naive Bayes we were training a **data model** that would allow us to **generate samples** given the class

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \ p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k).$$

So we were **modeling the data**.

However, there is a family of models that are trained to predict this (exactly what we want classifier to do):

$$p(C_k \mid \mathbf{x})$$

They are focused on determining which features are the best to **separate the classes**

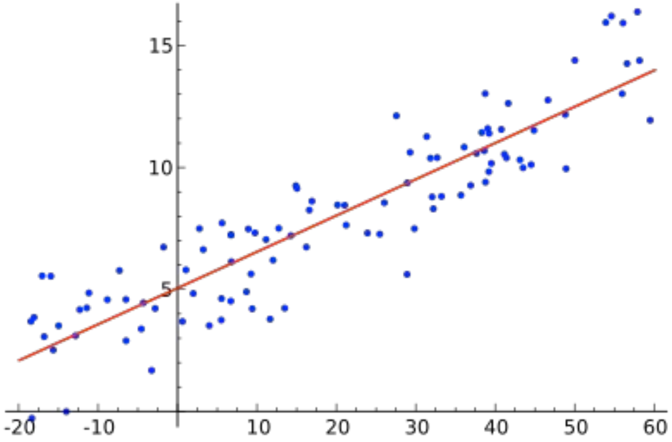
Plan

1. ~~Motivation~~
2. ~~Classification task~~
3. ~~Classifier example: Naive Bayes~~
4. ~~Classification quality evaluation~~
5. ~~Classification methods review~~
   a. Linear methods
   b. Metric methods
   c. Logical methods
   d. Ensembles
6. Typical tasks and special cases

Linear models

Usually they look like this

$$? = \sum_{i=1}^{N} w_i f_i$$



where **f** are features
(e.g., bag of ngrams),
and **w** are the weights we are to find

Training the linear regression so that it would return the conditional **probability** of the class given the data is not really possible :)

Logistic regression

Let's try to fix it by making the outputs

- nonnegative

$$p(c|x) = \frac{1}{Z}\exp\left(\sum_i w_i f_i(c,x)\right)$$

- lying between 0 and 1

$$p(c|x) = \frac{\exp\left(\sum_{i=1}^{N} w_i f_i(c,x)\right)}{\sum_{c' \in C} \exp\left(\sum_{i=1}^{N} w_i f_i(c',x)\right)}$$

Logistic regression: predictions

Can be solved in a way similar to a linear regression

$$
\begin{aligned}
\hat{c} &= \operatorname*{argmax}_{c \in C} P(c|x) \\
&= \operatorname*{argmax}_{c \in C} \frac{\exp\left(\sum_{i=1}^{N} w_i f_i(c,x)\right)}{\sum_{c' \in C} \exp\left(\sum_{i=1}^{N} w_i f_i(c',x)\right)} \\
&= \operatorname*{argmax}_{c \in C} \exp \sum_{i=1}^{N} w_i f_i(c,x) \\
&= \operatorname*{argmax}_{c \in C} \sum_{i=1}^{N} w_i f_i(c,x)
\end{aligned}
$$

Logistic regression: training

Maximizing conditional probability of the class (y) given the data (x)

$$\hat{w} = \underset{w}{\operatorname{argmax}} \log P(y^{(j)}|x^{(j)})$$

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_j \log P(y^{(j)}|x^{(j)})$$

$$L(w) = \sum_j \log P(y^{(j)}|x^{(j)}) = \sum_j \log \exp\left(\sum_{i=1}^{N} w_i f_i(y^{(j)}, x^{(j)})\right) - \sum_j \log \sum_{y' \in Y} \exp\left(\sum_{i=1}^{N} w_i f_i(y'^{(j)}, x^{(j)})\right)$$

For gradient ascend we need a derivative

$$L'(w) = \sum_j f_k(y^{(j)}, x^{(j)}) - \sum_j \sum_{y' \in Y} P(y'|x^{(j)}) f_k(y'^{(j)}, x^{(j)})$$

data-based feature counter

predicted feature values

Important problem: overfitting

Machine learning models can fit the training set ‘too well’: e.g. features values that occur only with one class label are a strong signal for the classifier (even if the number of such cases is not large)!

For example, logistic regression can assign a large weight to a particular feature $\mathbf{w}_i$

However, such cases may be too specific and this may not be a good rule when using the model in the wild!

“Models fitting too specific cases” usually fail to generalize. This is called **overfitting**.

Logistic regression: regularization

One way to fight overfitting is regularization: adding extra constraints to the task or restricting the possible solutions family

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_{j} \log P(y^{(j)} | x^{(j)}) - \alpha R(w)$$

L2-regularization
aka **shrinkage**
aka **Tikhonov's regularization**

$$R(W) = ||W||_2^2 = \sum_{j=1}^{N} w_j^2$$

...doesn't allow the weights to grow

Logistic regression: regularization

One way to fight overfitting is regularization: adding extra constraints to the task or restricting the possible solutions family

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_j \log P(y^{(j)}|x^{(j)}) - \alpha R(w)$$

L1-regularization
aka **LASSO** (least absolute shrinkage and selection operator)

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_j \log P(y^{(j)}|x^{(j)}) - \alpha \sum_{i=1}^{N} |w_i|$$

...doesn't just make the weights smaller but also allows to turn them into zero

Logistic regression: discussion

- \+ training and prediction is fast
- \+ more robust then naive Bayes and works better with correlated features

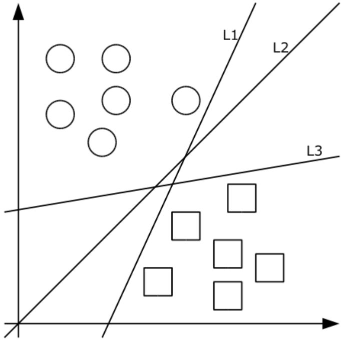
- \- has to be done: tuning regularization, feature normalization, feature selection
- \- probabilities estimates may not reflect the data, see

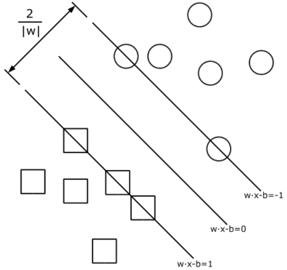
Linear models: SVM (Support Vector Machine)

Linear models usually build a separating hyperplane: different classes should be at different sides of it

Now let’s try to build a hyperplane so that the objects with different labels are at max. distance from it

This should help to generalize and be more confident when predicting classes





Вапник В. Н., Червоненкис А. Я. Теория распознавания образов. — М.: Наука, 1974.
Cortes C., Vapnik V. Support-vector networks // Machine Learning. — 1995. — Vol. 20, no. 3. — Pp. 273–297.

Linear models: SVM

Points of classes **c** from the set { -1, 1 }:

$$\{(\mathbf{x}_1, c_1), (\mathbf{x}_2, c_2), \ldots, (\mathbf{x}_n, c_n)\}$$

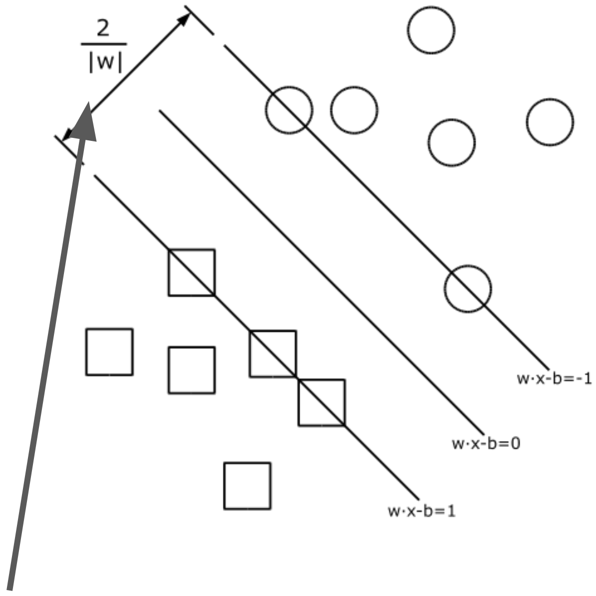
Separating hyperplane:

$$\mathbf{w} \cdot \mathbf{x} - b = 0.$$

two **parallel hyperplanes** that we can move without touching the samples in the case of linear separability:

$$\mathbf{w} \cdot \mathbf{x} - b = 1, \quad \mathbf{w} \cdot \mathbf{x} - b = -1.$$

So we minimize **|w|**, so that the distance between them was greater



Linear models: SVM

Quadratic programming task

$$\begin{cases} \|\mathbf{w}\|^2 \to \min \\ c_i(\mathbf{w} \cdot \mathbf{x_i} - b) \geq 1, \quad 1 \leq i \leq n. \end{cases}$$

with a few transformations we can reformulate the task like this:

$$\begin{cases} -\mathbf{L}(\lambda) = -\sum_{i=1}^{n} \lambda_\mathbf{i} + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_\mathbf{i} \lambda_\mathbf{j} c_i c_j (\mathbf{x_i} \cdot \mathbf{x_j}) \to \min_\lambda \\ \lambda_\mathbf{i} \geq 0, \quad 1 \leq i \leq n \\ \sum_{i=1}^{n} \lambda_\mathbf{i} c_i = 0 \end{cases}$$

this quadratic programming task has just one solution, which can be effectively found in the case if hundreds of thousands objects

Linear models: SVM

- there is an modification for multiple linearly inseparable classes

- take a look at the formulae at the previous slide: the features are used **only in the scalar product**

  hence we can redefine it; this way we'll move objects into the space of higher dimensionality where they may be linearly separable

  this is called the **kernel trick**

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^d$$

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + 1)^d$$

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$$

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

$$k(\mathbf{x}, \mathbf{x}') = \tanh(\kappa \mathbf{x} \cdot \mathbf{x}' + c)$$

SVM, discussion

+ separating hyperplanes with margin usually deliver a more ‘confident’ solution
+ the optimization task has effective solution methods

- not robust to outliers (those that are close to the separation hyperplane)
- choosing the kernel is black magic; common sense doesn’t always work
- when there is no prior belief in linear separability of the classes, one has to tune parameters

Plan

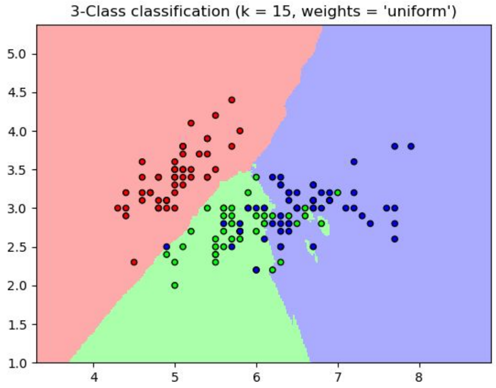
1. ~~Motivation~~
2. ~~Classification task~~
3. ~~Classifier example: Naive Bayes~~
4. ~~Classification quality evaluation~~
5. ~~Classification methods review~~
    a. ~~Linear methods~~
    b. Metric methods
    c. Logical methods
    d. Ensembles
6. Typical tasks and special cases

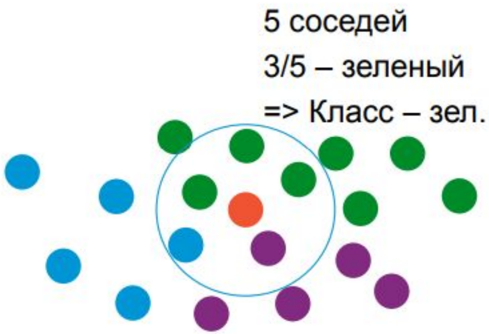
Metric classifiers: kNN

At the core -- **compactness hypothesis**: objects that are close to each other in a metric space should have the same label

k Nearest Neighbours method**:** no training, a classified object is given the most popular label among **k** closest objects in the train set

The larger the **k**, the more smooth are the borders between classes; however, if the k is too large, **underfitting** is possible





http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classification-p

https://compscicenter.ru/media/slides/nlp_2014_spring/2014_03_03_nlp_2014_spring.pdf

kNN: how to improve

One can

- **use the order of the neighbours (when sorted by distance)**
  as a ‘vote weight’ (the closer, the more important is the label vote)

- **use neighbours distances** to the classified object
  (vote weight is set by function, take a look at the Parzen window method)

- **filter** a set of representative objects in the training set
  (predictions are made faster + removing outliers helps)

kNN: discussion

+ non-linear, classes samples groups can be of arbitrary form and shape
+ a natural way to do the multiclass classification

- may be too expensive to store and use for predictions all/representative training set objects
- depends on the training set too much
- [usually unsuitable for large dimensions]

Plan

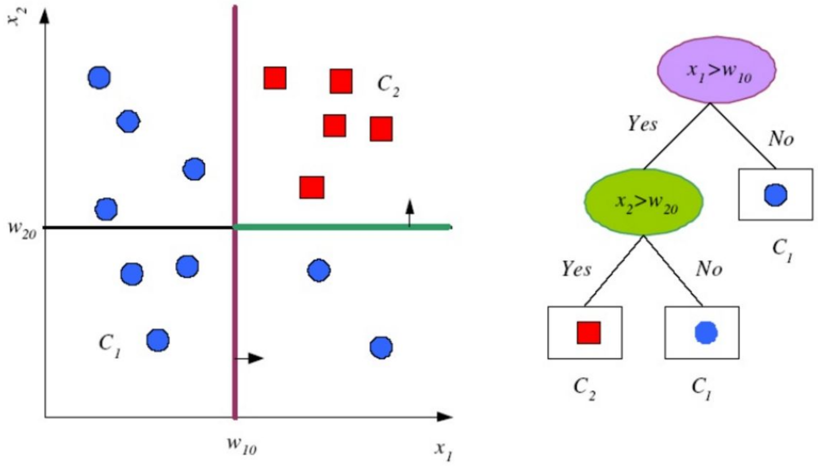
1. ~~Motivation~~
2. ~~Classification task~~
3. ~~Classifier example: Naive Bayes~~
4. ~~Classification quality evaluation~~
5. ~~Classification methods review~~
    a. ~~Linear methods~~
    b. ~~Metric methods~~
    c. Logical methods
    d. Ensembles
6. Typical tasks and special cases

Logical classifiers: decision trees

We build a structure setting the conditions splitting the data

For every classified object we go through that structure (tree) checking the conditions on the way (e.g. “is there a word **genome** in the text?”) from top to bottom, taking the label in the leaf as a result

The samples space is partitioned into the parallelograms, one label is set to each



https://www.slideshare.net/marinasantini1/lecture02-machine-learning

Logical classifiers: decision trees

ID3 algorithm

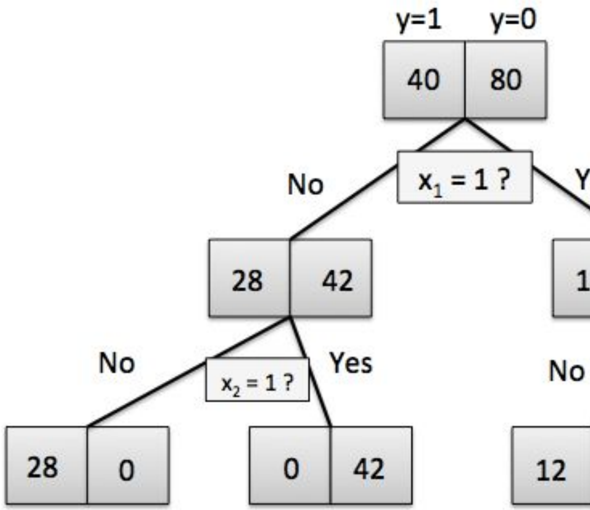
How “impure” the distribution of classes in S is is characterized by the entropy over shares of the samples of different labels

$$H(S) = \sum_{x \in X} -p(x) \log_2 p(x)$$

1. For every feature **A** and for every possible data partitioning **T** by it compute the **information gain**

$$IG(A, S) = H(S) - \sum_{t \in T} p(t) H(t)$$

2. Split the dataset using the feature and the partitioning with the MAX **IG**
3. Do 1-2 recursively with the subsets until there are no more samples or until **IG** stops to grow



Logical classifiers: decision trees

Classical algorithms: ID3, C4.5, CART, …

A few heuristics to fight with overfitting

E.g. **pruning**: we replace the subtree with a leaf with the most frequent label in the former subtree if that doesn't hurt the quality of predictions on the **dev set**

Decision trees: discussion

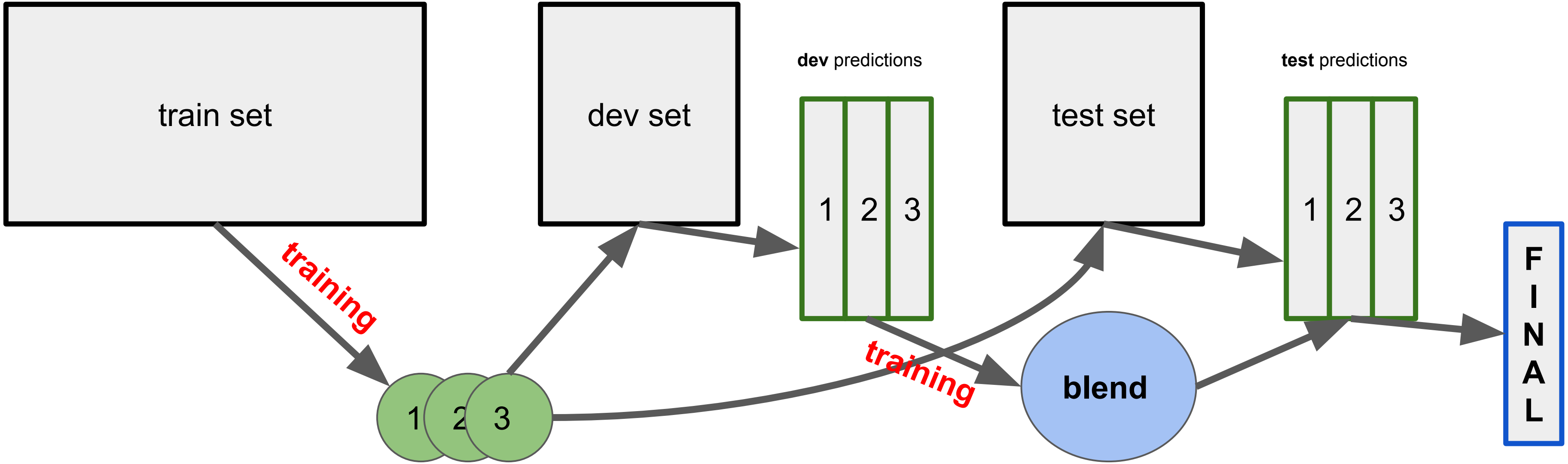
+ easy to interpret
+ don’t have many assumptions on what the solution should look like

- overfit easily
- not that great for large dimensions

Plan

1. ~~Motivation~~
2. ~~Classification task~~
3. ~~Classifier example: Naive Bayes~~
4. ~~Classification quality evaluation~~
5. ~~Classification methods review~~
    a. ~~Linear methods~~
    b. ~~Metric methods~~
    c. ~~Logical methods~~
    d. Ensembles
6. Typical tasks and special cases

Machine learning models ensembles: blending



We train at **train** set, we tune weights at **dev**, we check quality at **test**

- if you don't tune too hard, this helps to overcome overfitting
- main advantage: **quick and dirty**; the first thing to try

Machine learning models ensembles

Using multiple models for predictions may help

- not to ovefit
- to get a more rich solutions space than of any of the models the ensemble is composed of

Blending: joining the predictions of multiple models into one

- if we have probabilities, we can take the **weighted sum**
- weights for the linear combination may be trained + we can even train a model over predictions (BUT: overfitting alert!)
- if we predict classes, one can take a mode of the predicted labels

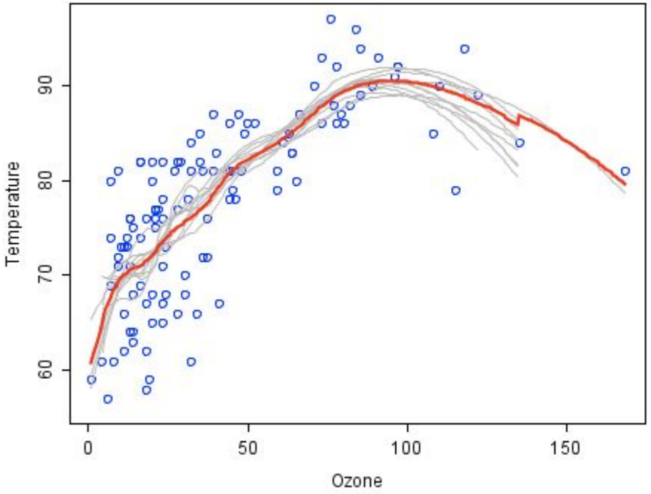
Machine learning models ensembles: bagging

Bagging (**b**ootstrap **agg**regat**ing**) — sampling a few datasets from the training set, training classifiers on the independently

Feature bagging (attribute bagging, random subspace method) — sampling **subsets of features** and training classifiers on such sets independently

The resulting model is a consensus or a weighted vote

This allows for being more confident in predictions and helps overcome overfitting



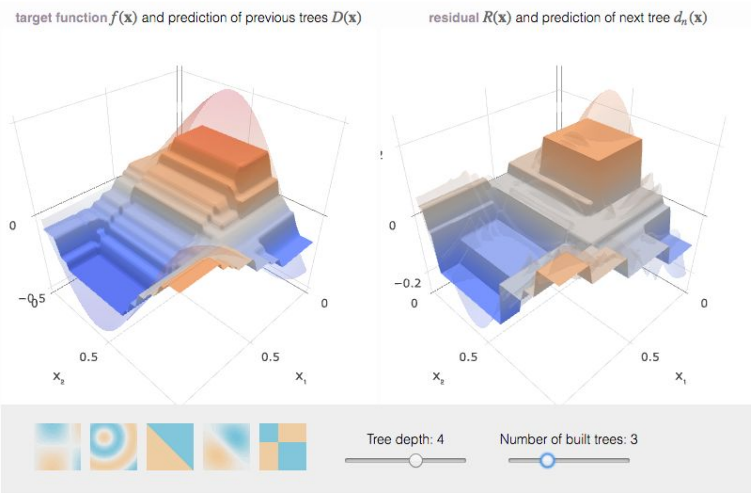
https://en.wikipedia.org/wiki/Bootstrap_aggregating

Machine learning models ensembles: boosting

The core idea is to use a bunch of weak classifiers (non-random though) to build a strong one

Usually done like this:

1) incrementally training weak classifiers
2) when adding each of them we increase the weight of previously wrongly classified samples
3) classifiers are added into the composition with the weight reflecting the quality they've shown



Cool demo

More stuff one needs to know

- **other ways to measure quality**, e.g., comparison with random predictions
- **feature selection** (PMI, DIA, Chi-square, ...)
- how to deal with label-**imbalanced datasets**
- how to deal with **small training data**
- **tuning hyperparameters** methods
  (grid search, random search, bayesian optimization,
  gradient-based optimization)
- ...

Important special cases

Sentiment analysis: building ‘sentimental words’ vocabularies, e.g.

- semi-automatic (given initial sentimental seed words)
- custom vocabulary building, e.g. for specific domains

Topic classification:

- topic hierarchy building; the less supervision there is, the better
- dealing with the case where there is no true topic in label list yet

Tools and instruments

Models zoos to give each a try:

- Weka (GUI)
- Scikit-Learn
- Mallet

Text classifiers can be implemented using:
nltk, spaCy, H2O, mllib, Vowpal Wabbit, BigARTM, …

Standard datasets for English:

- 20 Newsgroups (18k posts; 20 topics)
- Reuters Newswire Topic Classification (Reuters-21578; topical categories)
- IMDB Movie Review Sentiment Classification (stanford; sentiment)
- News Group Movie Review Sentiment Classification (cornell; sentiment)

Datasets are also many, any colour you like

Used/recommended literature

1. Yandex Data School course on machine learning + similar lecture notes: this
2. The Elements of Statistical Learning and other classical books on machine learning (classification is everywhere)
3. Martin/Jurafsky, Chapters 6-7 in Ed. 3
4. Intro into IR (NB, kNN, Rocchio, SVM,...)
5. Wikipedia
6. CSC lectures, 2014 [Russian]

Text classification

Anton Alekseev
Steklov Mathematical Institute in St Petersburg

NRU ITMO, St Petersburg, 2018
anton.m.alexeyev+itmo@gmail.com

This time thanks go to Denis Kiryanov, Semyon Danilov, Viktor Evstratov