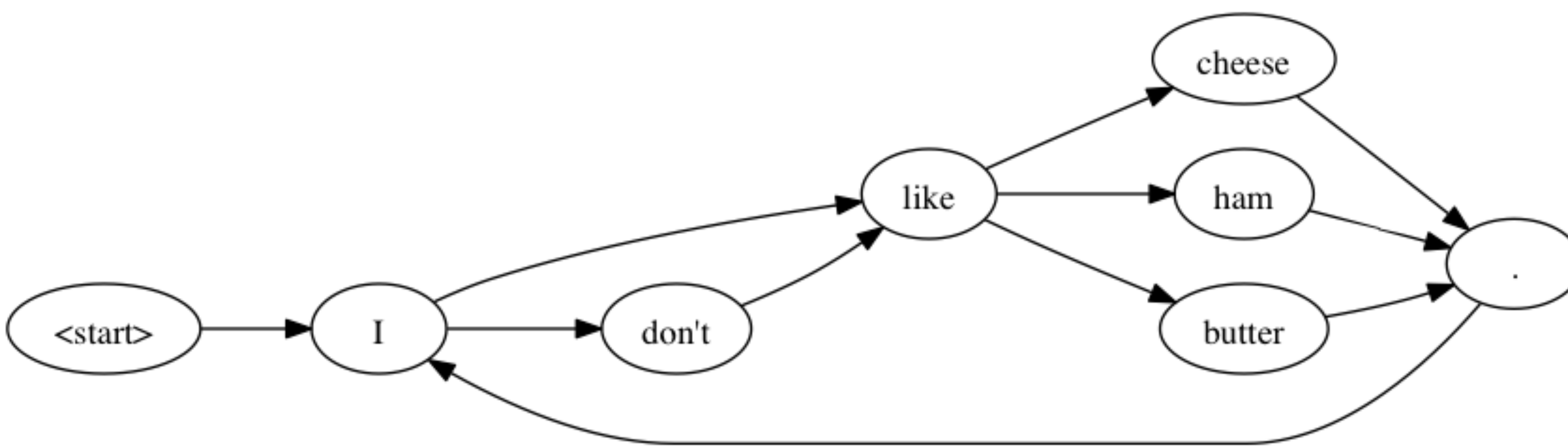
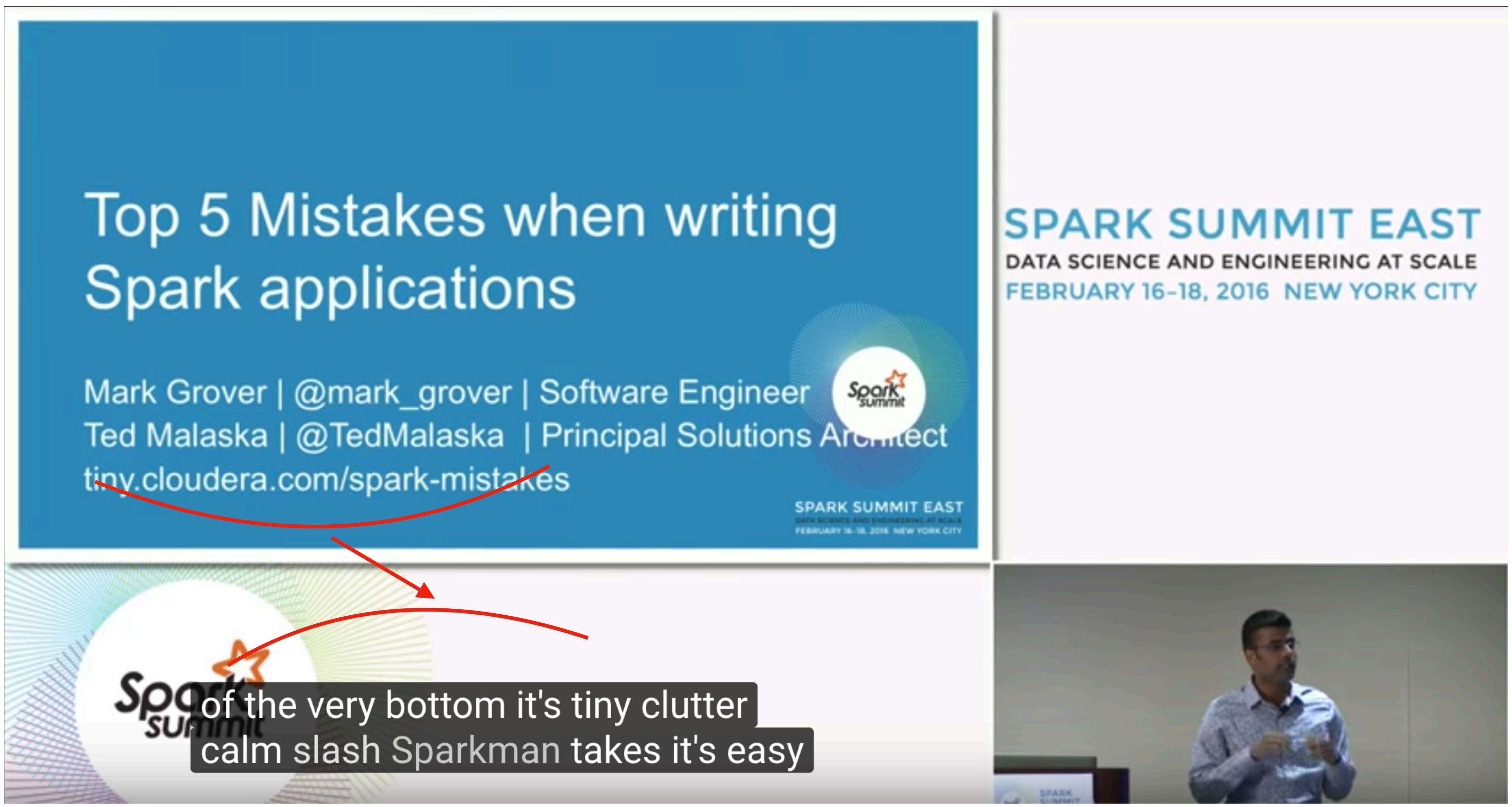
Language models

Eugeny Malyutin



Motivation:

There are a lot of tasks to estimate whether text is «natural» or «comprehensible».
Sometimes a clever way to estimate the word sequence probability is enough



Possible tasks:

- **Machine translation:**
  他 向 记者 介绍了 主要 内容
  He to reporters introduced main content

  As part of the process we might have built the following set of potential rough English translations:
  he introduced reporters to the main contents of the statement
  he briefed to reporters the main contents of the statement
  he briefed reporters on the main contents of the statement

What do you prefer?

Possible tasks:

- **Machine translation:**
  他 向 记者 介绍了 主要 内容
  He to reporters introduced main content

  As part of the process we might have built the following set of potential rough English translations:
  he introduced reporters to the main contents of the statement
  he briefed to reporters the main contents of the statement
  he briefed reporters on the main contents of the statement

Another tasks:

- **Speech recognition / machine translation / spelling correction / augmentative communication:**
  e.g.: having generated several possible decodings of the phrase, one has to choose ‘the most probable’ (from the language’s point of view)

- **Information retrieval:**
  ranking: for every document **d** we build ‘its language model’ and sort all documents by **P(q|d)** (where **q** is a query)

- **Text generation (spam also):**
  text generators, imitating the provided text collection’s style: Blabla-bots, spam, science articles, linux core C++ code (**http://karpathy.github.io/2015/05/21/rnn-effectiveness/**)
  //this blogue post is not about language models but about fun.

Another tasks:

- **Speech recognition / machine translation / spelling correction / augmentative communication:**
  e.g.: having generated several possible decodings of the phrase, one has to choose ‘the most probable’ (from the language’s point of view)

- **Information retrieval:**
  ranking: for every document **d** we build ‘its language model’ and sort all documents by **P(q|d)** (where **q** is a query)

- **Text generation (spam also):**
  text generators, imitating the provided text collection’s style: Blabla-bots, spam, science articles, linux core C++ code (**http://karpathy.github.io/2015/05/21/rnn-effectiveness/**)
  //this blogue post is not about language models but about fun.

Intuition:

Hello, how are … ?

Intuition:

Hello, how are … ?

on? are?

Intuition:

Hello, how are … ?

you

Language model allows us to estimate the probability of any sequence of words (alternative formulation: to estimate the probability of the next word) // Why are these formulations equivalent?.

Intuition:

Conditional probability:

$$P(Y|X) = \frac{P(X,Y)}{P(Y)} \Rightarrow P(X,Y) = P(Y|X)P(X)$$

Chain rule for greater number of variables:

$$P(x_1 x_2 \ldots x_n) = P(x_n | x_1 x_2 \, . \, . \, x_{n-1}) \, . \, . \, P(x_2 | x_1) P(x_1)$$

Can we compute it?

$$P(x_i | x_1 x_2 \, . \, . \, x_{i-1}) = \frac{Count(x_1 x_2 \, . \, . \, x_{i-1} x_i)}{Count(x_1 x_2 \, . \, . \, x_{i-1})}$$

Intuition:

Can we compute it?

$$P(x_i | x_1 x_2 .. x_{i-1}) = \frac{Count(x_1 x_2 .. x_{i-1} x_i)}{Count(x_1 x_2 .. x_{i-1})}$$

Sharp corners:

- Not enough data
- Long nGrams -> totally-predictable text parts -> nothing new; (some kind of overfitting)

Any ideas?

Intuition:

Can we compute it?

$$P(x_i \,|\, x_1 x_2 \,..\, x_{i-1}) = \frac{Count(x_1 x_2 \,..\, x_{i-1} x_i)}{Count(x_1 x_2 \,..\, x_{i-1})}$$

Any ideas?

Markov assumption: $P(x_i \,|\, x_1 \,..\, x_{i-1}) = P(x_i \,|\, x_{i-K} \,..\, x_{i-1})$

…means that current event depends on not more then K preceding one;

Example:

- **Unigram:** P(imagine there is no heaven) = P(imagine) x P(there) x … P(heaven)
- **Bigram:** P(imagine there is no heaven) = P(heaven | no) x P(no|is) x P(is|there)…

Intuition:

Can we compute it?

$$P(x_i \,|\, x_1 x_2 \ldots x_{i-1}) = \frac{Count(x_1 x_2 \ldots x_{i-1} x_i)}{Count(x_1 x_2 \ldots x_{i-1})}$$

Any ideas?

Markov assumption: $P(x_i \,|\, x_1 \ldots x_{i-1}) = P(x_i \,|\, x_{i-K} \ldots x_{i-1})$

…means that current event depends on not more then K preceding one;

Example:

- **Unigram:** P(imagine there is no heaven) = P(imagine) x P(there) x … P(heaven)
- **Bigram:** P(imagine there is no heaven) = P(heaven | no) x P(no|is) x P(is|there)…

Practical tricks:

- Most popular - trigram models; Sometimes it's OK to use 4 or 5-grams models when there is sufficient training data;
- We will use <s> as «sentence start» and </s> as «sentence end» symbols to keep 2-grammity for first and last words.
  For trigram models sentences starts with <s><s> symbols.
- Log probabilities:
  - P(x) < 1 => P_1 * P_2 * … * P_n <<< 0
  - log(x1 * x2) = log(x1) + log(x2)
  - p1 * p2 * p3 * p4 = exp(log(p1) + log(p2) + log(p3) + log(p4))

Berkeley Restaurant Project:

- Berkeley Restaurant Project - a dialogue system from the last century that answered questions about a database of restaurants in Berkeley;

- can you tell me about any good cantonese restaurants close by
  mid priced thai food is what i'm looking for
  tell me about chez panisse
  can you give me a listing of the kinds of food that are available
  i'm looking for a good place to eat breakfast
  when is caffe venezia open during the day

	i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Berkeley Restaurant Project:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

- P(i|<s>) = 0.25   P(english|want) = 0.0011 (additional data)
  P(food|english) = 0.5   P(</s>|food) = 0.68
- **Can you compute** P(i want english food) ?

Berkeley Restaurant Project:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

- P(i|<s>) = 0.25 P(english|want) = 0.0011 (additional data)
  P(food|english) = 0.5 P(</s>|food) = 0.68

- P(i want english food) = P(i|<s>) x P(want|i) x P(english|want) x P(food|english) x P(</s>|food)
  = 0.25 × 0.33 × 0.011× 0.5 ×0.68
  = .000031

Berkeley Restaurant Project:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

- P(i|<s>) = 0.25   P(english|want) = 0.0011 (additional data)
  P(food|english) = 0.5   P(</s>|food) = 0.68

- P(i want english food) = P(i|<s>) x P(want|i) x P(english|want) x P(food|english) x P(</s>|food)
  = 0.25 × 0.33 × 0.011× 0.5 ×0.68
  = .000031

Model evaluation:

- **Extrinsic**
  Checking quality by inducing the model into a bigger useful task (machine translation, spelling correction, ...). If the target metric (where the money is: translators work time, editor’s time, clicks count, earned money, etc.) goes up, the model has become better

- **Intrinsic**
  ~~Evaluation for the poor~~ we need estimates when extrinsic evaluation is too expensive or when one doesn’t want the results to be related to some specific application (if the model is universal to certain extent); also a metric that shows us how ‘good’ the model is

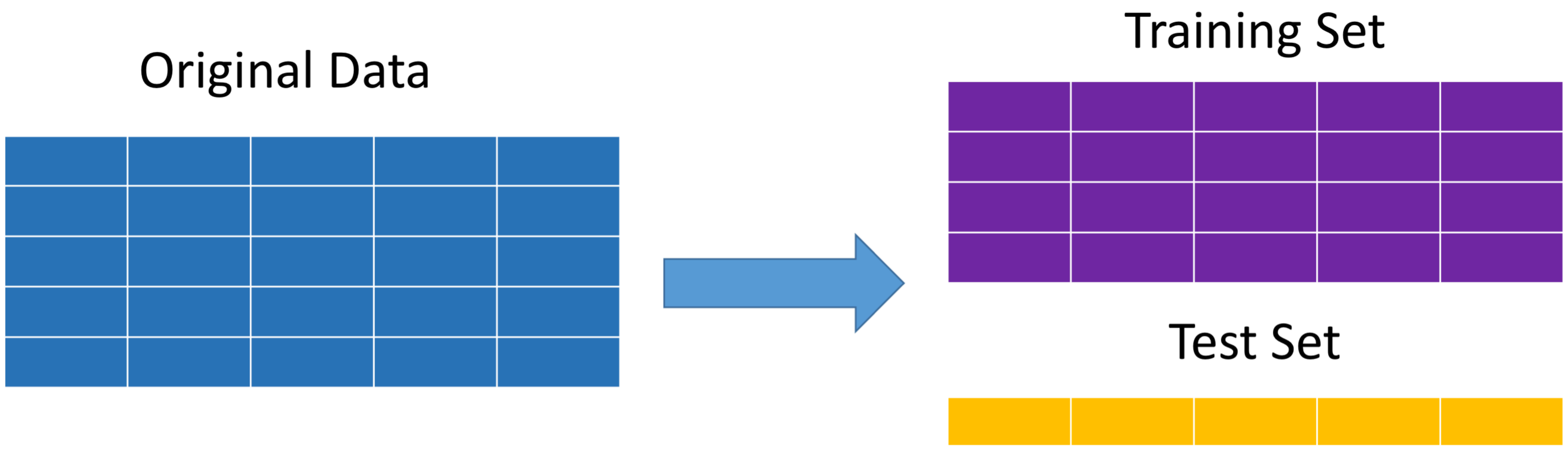
Quality evaluation:

We have the data, we have the metric

We split the data into:

- train set (for tuning models)
- test set (for trained models evaluation)

We have to believe that train and test set data samples are from “the same distribution” (otherwise we won’t be able to train anything useful)



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?

Quality evaluation:

- K-fold scheme:
- **Train** - training model
- **Validation** - evaluating quality + analyzing errors + tuning hyperparameters
- **Holdout** - blind quality evaluation: looking at quality metric ONLY + not too often, so as not to overfit

Perplexity:

- The large the probability of the test text, the closer the model is to life

- Perplexity — inverse probability of the text normalized by word sequence length:

$$PP(W) = P(x_1 \ldots x_N)^{\frac{1}{N}}$$

Is is evident the less is better;

- Looks like entropy:

$$PP(W) = P(x_1 \ldots x_N)^{\frac{1}{N}} = e^{-\frac{1}{N}\sum_{i=1}^{N} log(P(x_i|x_1..x_{i-1}))}$$

Quality evaluation: example

- Training on 38M tokens
- Testing on 1.5M
- Dataset: Wall Street Journal

	1-gram	2-gram	3-gram
Perplexity	962	170	109

Berkeley Restaurant Project:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

- P(i|<s>) = 0.25 P(english|want) = 0.0011 (additional data) P(food|english) = 0.5 P(</s>|food) = 0.68
- **P(i want english food)** = P(i|<s>) x P(want|i) x P(english|want) x P(food|english) x P(</s>|food)
  = 0.25 × 0.33 × 0.011× 0.5 ×0.68
  = .000031
- $$PP(W) = P(x_1 \ldots x_N)^{\frac{1}{N}} = e^{-\frac{1}{N}\sum_{i=1}^{N} log(P(x_i|x_1..x_{i-1}))}$$
- **Any problems?**

Generalization capability discussion:

- There is no such perfect corpus where all possible n-grams occur at least once!

- The model we have described returns P(x,...) = 0 when run on the text that contains at least one ngram that was not present in train set

- Evident enough, the model must generalize (and not just encode with non-zeros what was present in the train set)

  a very natural solution is to **convert zeros to small values** (a-k-a «reserve some probability for words we never met before)»

Unknown words:

1st approach (encode it):

- **Choose** a vocabulary (word list) that is fixed in advance.
- **Convert** in the training set any word that is not in this set (any OOV word) to the unknown word token in a text normalization step.
- **Estimate** the probabilities for from its counts just like any other regular word in the training set.

2nd approach (no prior vocabulary)

- Choose K (or threshold f_min)
- Any words not from top-K by relative frequency (or with freq < f_min) -> encode with <UNK>

Laplacian smoothing:

- So,

$$P(w_i \mid w_{i-1}) = \frac{Count(w_i, w_{i-1}) + 1}{Count(w_i) + V}$$

- If we sum over w_i - we'll see V should be the cardinality of unigrams set, otherwise P couldn't be called provability

- Doesn't work well
(too much useful weight is transferred to zero)

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Probabilities for BRC with Laplacian smoothing

Laplacian smoothing:

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Recovered frequency table for BRC

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Original frequency table for BRC

Laplacian smoothing:

- Too hard for real language modeling!
- Useful for language classification task based on unigrams
- There is one fix for the Poor:

$$P(w_i \mid w_{i-1}) = \frac{Count(w_i, w_{i-1}) + \alpha}{Count(w_i) + \alpha V}$$

Smoothing

- To be continued…