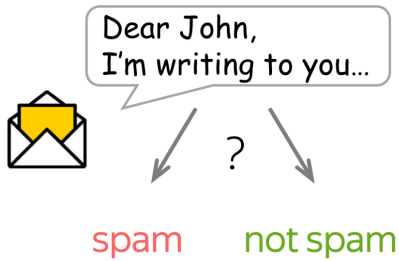
Text Classification

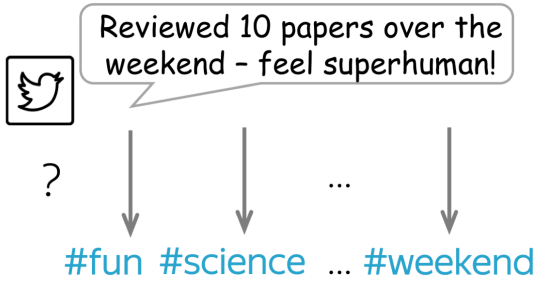
based on https://lena-voita.github.io/nlp_course/text_classification.html

Text classification is an extremely popular task. You enjoy working text classifiers in your mail agent: it classifies letters and filters spam. Other applications include document classification, review classification, etc.

Text classifiers are often used not as an individual task, but as part of bigger pipelines. For example, a voice assistant classifies your utterance to understand what you want (e.g., set the alarm, order a taxi or just chat) and passes your message to different models depending on the classifier's decision. Another example is a web search engine: it can use classifiers to identify the query language, to predict the type of your query (e.g., informational, navigational, transactional), to understand whether you what to see pictures or video in addition to documents, etc.

Since most of the classification datasets assume that only one label is correct (you will see this right now!), in the lecture we deal with this type of classification, i.e. the single-label classification. We mention multi-label classification in a separate section (Multi-Label Classification).





Datasets for Classification

Datasets for text classification are very different in terms of size (both dataset size and examples' size), what is classified, and the number of labels. Look at the statistics below.

Dataset	Type	Number of labels	Size (train/test)	Avg. length (tokens)
SST	sentiment	5 or 2	8.5k / 1.1k	19
IMDb Review	sentiment	2	25k / 25k	271
Yelp Review	sentiment	5 or 2	650k / 50k	179
Amazon Review	sentiment	5 or 2	3m / 650k	79
TREC	question	6	5.5k / 0.5k	10
Yahoo! Answers	question	10	1.4m / 60k	131
AG's News	topic	4	120k / 7.6k	44
Sogou News	topic	6	54k / 6k	737
DBPedia	topic	14	560k / 70k	67

Some of the datasets can be downloaded here.

The most popular datasets are for sentiment classification. They consist of reviews of movies, places or restaurants, and products. There are also datasets for question type classification and topic classification.

To better understand typical classification tasks, below you can look at the examples from different datasets.

General View

Here we provide a general view on classification and introduce the notation. This section applies to both classical and neural approaches.

We assume that we have a collection of documents with ground-truth labels. The input of a classifier is a document $(x = (x_1, \ldots, x_n))$ with tokens $((x_1, \ldots, x_n))$, the output is a label $(y \in 1 \ldots k)$. Usually, a classifier estimates probability distribution over classes, and we want the probability of the correct class to be the highest.

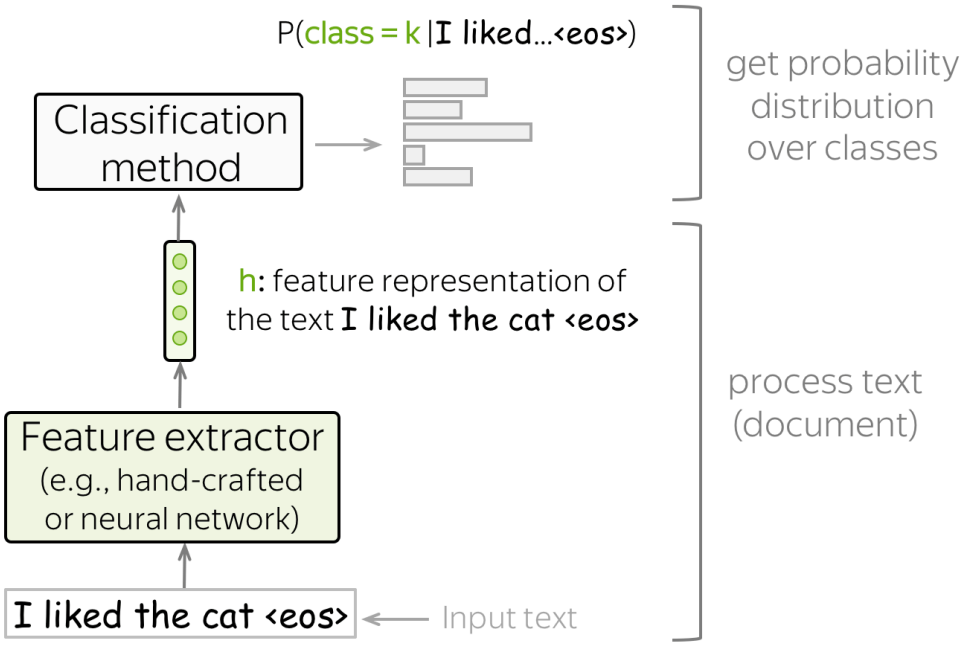
Get Feature Representation and Classify

Text classifiers have the following structure:

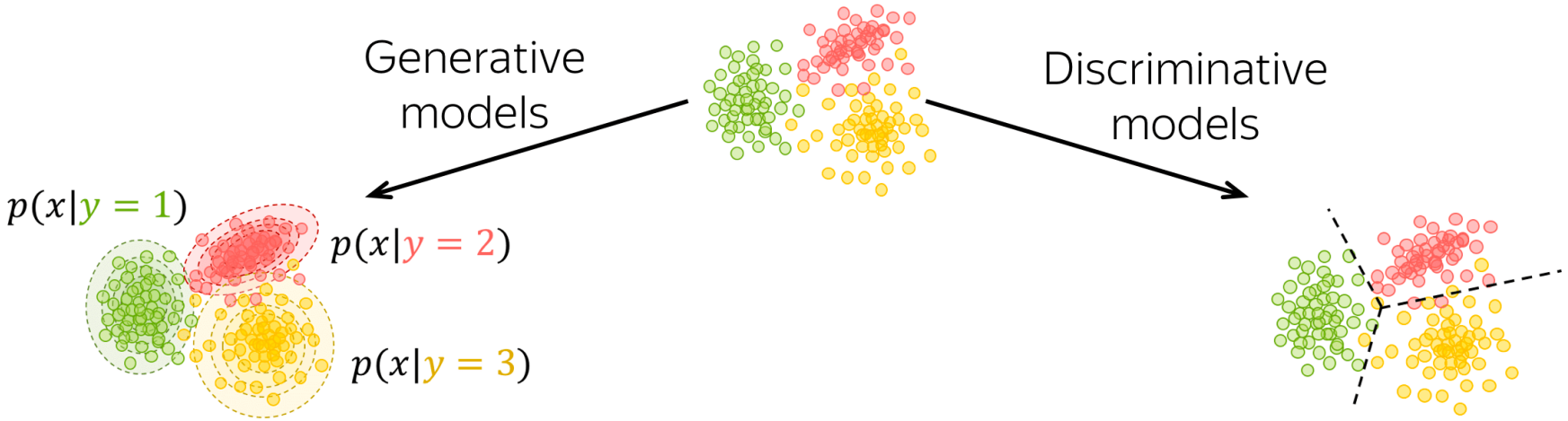
- feature extractor
  A feature extractor can be either manually defined (as in classical approaches) or learned (e.g., with neural networks).

- classifier
  A classifier has to assign class probabilities given feature representation of a text. The most common way to do this is using logistic regression, but other variants are also possible (e.g., Naive Bayes classifier or SVM).

In this lecture, we'll mostly be looking at different ways to build feature representation of a text and to use this representation to get class probabilities.



Generative (Joint) and Discriminative (Conditional) Models



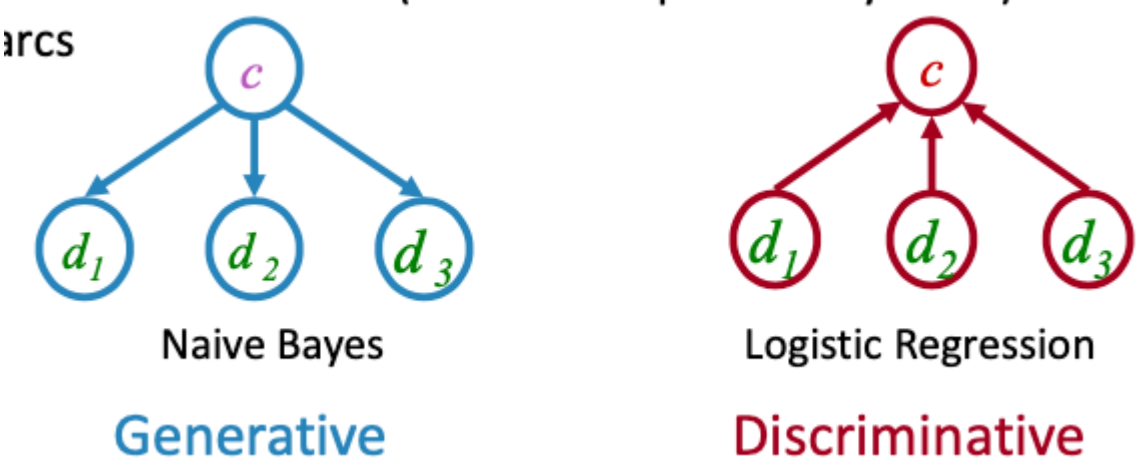
Learn: data distribution $p(x, y) = p(x|y) \cdot p(y)$

How predict: $y = \arg\max_k P(x, y = k) =$

$= \arg\max_k P(x|y = k) \cdot P(y = k)$

Learn: boundary between classes $p(y|x)$

How predict: $y = \arg\max_k P(y = k|x)$



A classification model can be either generative or discriminative.

- generative models
  Generative models learn joint probability distribution of data $(p(x, y) = p(x|y) \cdot p(y))$. To make a prediction given an input x, these models pick a class with the highest joint probability: $(y = \arg\max_k p(x|y = k) \cdot p(y = k))$.
- discriminative models
  Discriminative models are interested only in the conditional probability $(p(y|x))$, i.e. they learn only the border between classes. To make a prediction given an input x, these models pick a class with the highest conditional probability: $(y = \arg\max_k p(y = k|x))$.

In this lecture, we will meet both generative and discriminative models.

Classical Methods for Text Classification

In this part, we consider classical approaches for text classification. They were developed long before neural networks became popular, and for small datasets can still perform comparably to neural models.

Naive Bayes Classifier

A high-level idea of the Naive Bayes approach is given below: we rewrite the conditional class probability $P(y = k|x)$ using Bayes's rule and get $P(x|y = k) \cdot P(y = k)$.

Bayes' rule (hence Naïve Bayes)

Ignore P(x) – it does not influence the argmax

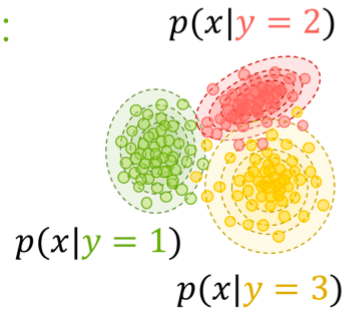
$$y^* = \arg\max_k P(y=k|x) = \arg\max_k \frac{P(x|y=k)\cdot P(y=k)}{P(x)} = \arg\max_k P(x|y=k)\cdot P(y=k)$$

need to define this

This is a generative model!

$$y^* = \arg\max_k \underbrace{P(y=k|x)}_{\text{posterior}} = \arg\max_k \underbrace{P(x|y=k)}_{\text{joint}} \cdot \underbrace{P(y=k)}_{\text{prior}} = \arg\max_k \underbrace{P(x,y=k)}_{\text{joint}}$$

posterior probability: after looking at data (i.e., we know **x**)



prior probability: before looking at data (i.e., we don't know **x**)

joint probability (hence the model is **generative**)

Naive Bayes is a generative model: it models the joint probability of data.

Note also the terminology:

- prior probability $P(y=k)$: class probability before looking at data (i.e., before knowing x;
- posterior probability $P(y=k|x)$: class probability after looking at data (i.e., after knowing the specific x;
- joint probability $P(x,y)$: the joint probability of data (i.e., both examples x and labels y;
- maximum a posteriori (MAP) estimate: we pick the class with the highest posterior probability.

How to define P(x|y=k) and P(y=k)?

P(y=k): count labels

$P(y = k)$ is very easy to get: we can just evaluate the proportion of documents with the label k (this is the maximum likelihood estimate, MLE). Namely,

$$P(y = k) = \frac{N(y = k)}{\sum_i N(y = i)},$$

where $N(y = k)$ is the number of examples (documents) with the label k.

P(x|y=k): use the "naive" assumptions, then count

Here we assume that document x is represented as a set of features, e.g., a set of its words $(x_1, \ldots, x_n)$:

$$P(x|y = k) = P(x_1, \ldots, x_n|y).$$

The Naive Bayes assumptions are

- Bag of Words assumption: word order does not matter,
- Conditional Independence assumption: features (words) are independent given the class.

Intuitively, we assume that the probability of each word to appear in a document with class k does not depend on context (neither word order nor other words at all). For example, we can say that awesome, brilliant, great are more likely to appear in documents with a positive sentiment and awful, boring, bad are more likely in negative documents, but we know nothing about how these (or other) words influence each other.

With these "naive" assumptions we get:

$$P(x|y = k) = P(x_1, \ldots, x_n|y) = \prod_{t=1}^{n} P(x_t|y = k).$$

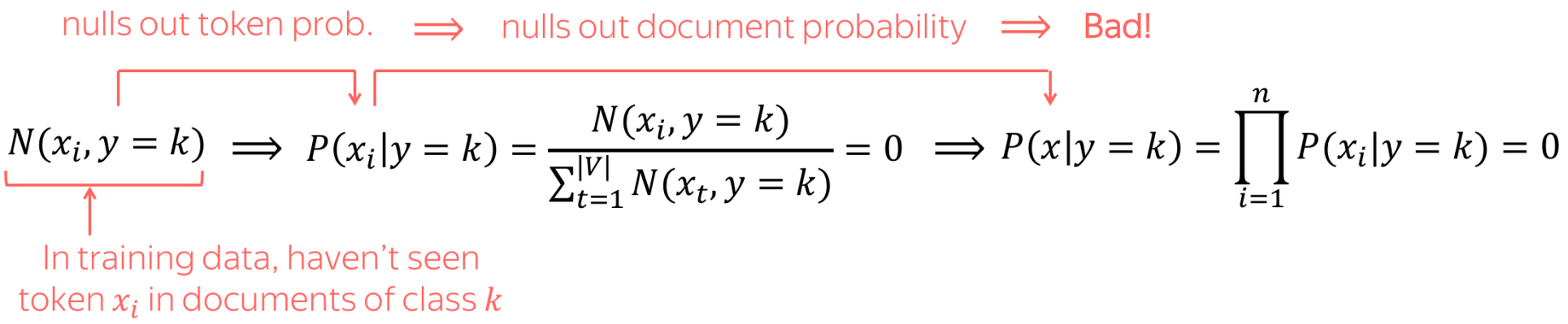
The probabilities $P(x_i|y = k)$ are estimated as the proportion of times the word x_i appeared in documents of class k among all tokens in these documents:

$$P(x_i|y = k) = \frac{N(x_i, y = k)}{\sum_{t=1}^{|V|} N(x_t, y = k)},$$

where $N(x_i, y = k)$ is the number of times the token x_i appeared in documents with the label k, V is the vocabulary (more generally, a set of all possible features).

What if $N(x_i, y = k) = 0$? Need to avoid this!

What if $N(x_i, y = k) = 0$, i.e. in training we haven't seen the token x_i in the documents with class k? This will null out the probability of the whole document, and this is not what we want! For example, if we haven't seen some rare words (e.g., pterodactyl or abracadabra) in training positive examples, it does not mean that a positive document can never contain these words.



To avoid this, we'll use a simple trick: we add to counts of all words a small δ:

$$P(x_i|y = k) = \frac{\delta + N(x_i, y = k)}{\sum_{t=1}^{|V|}(\delta + N(x_t, y = k))} = \frac{\delta + N(x_i, y = k)}{\delta \cdot |V| + \sum_{t=1}^{|V|} N(x_t, y = k)},$$

where δ can be chosen using cross-validation.

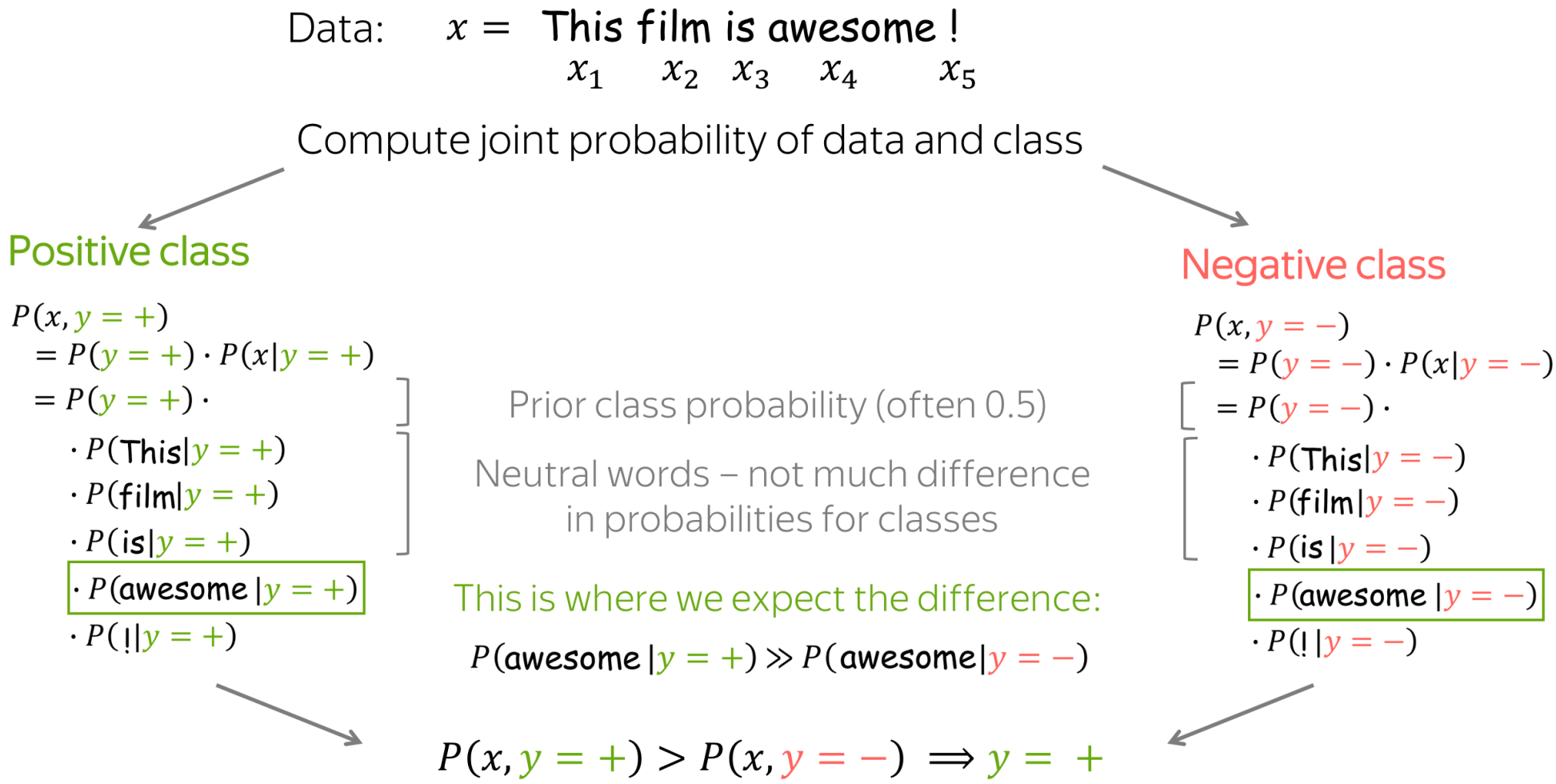
Note: this is Laplace smoothing (aka Add-1 smoothing if $\delta = 1$. We'll learn more about smoothings in the next lecture when talking about Language Modeling.

Making a Prediction

As we already mentioned, Naive Bayes (and, more broadly, generative models) make a prediction based on the joint probability of data and class:

$$y^* = \arg\max_k P(x, y = k) = \arg\max_k P(y = k) \cdot P(x|y = k).$$

Intuitively, Naive Bayes expects that some words serve as class indicators. For example, for sentiment classification tokens awesome, brilliant, great will have higher probability given positive class then negative. Similarly, tokens awful, boring, bad will have higher probability given negative class then positive.



Worked Sentiment Example with add-1 Smoothing

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable ~~with~~ no fun

1. Prior from training:

$$\hat{P}(c_j) = \frac{N_{c_j}}{N_{total}}$$

P(-) = 3/5

P(+) = 2/5

2. Drop "with"

3. Likelihoods from training:

$$p(w_i|c) = \frac{count(w_i, c) + 1}{(\sum_{w \in V} count(w, c)) + |V|}$$

$$P(\text{“predictable”}|-) = \frac{1+1}{14+20} \qquad P(\text{“predictable”}|+) = \frac{0+1}{9+20}$$

$$P(\text{“no”}|-) = \frac{1+1}{14+20} \qquad P(\text{“no”}|+) = \frac{0+1}{9+20}$$

$$P(\text{“fun”}|-) = \frac{0+1}{14+20} \qquad P(\text{“fun”}|+) = \frac{1+1}{9+20}$$

4. Scoring the test set:

$$P(-)P(S|-) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5}$$

$$P(+)P(S|+) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5}$$

Final Notes on Naive Bayes

Practical Note: Sum of Log-Probabilities Instead of Product of Probabilities

The main expression Naive Bayes uses for classification is a product lot of probabilities:

$$P(x, y = k) = P(y = k) \cdot P(x_1, \ldots, x_n|y) = P(y = k) \cdot \prod_{t=1}^{n} P(x_t|y = k).$$

A product of many probabilities may be very unstable numerically. Therefore, usually instead of $P(x, y)$ we consider $\log P(x, y)$:

$$\log P(x, y = k) = \log P(y = k) + \sum_{t=1}^{n} \log P(x_t | y = k).$$

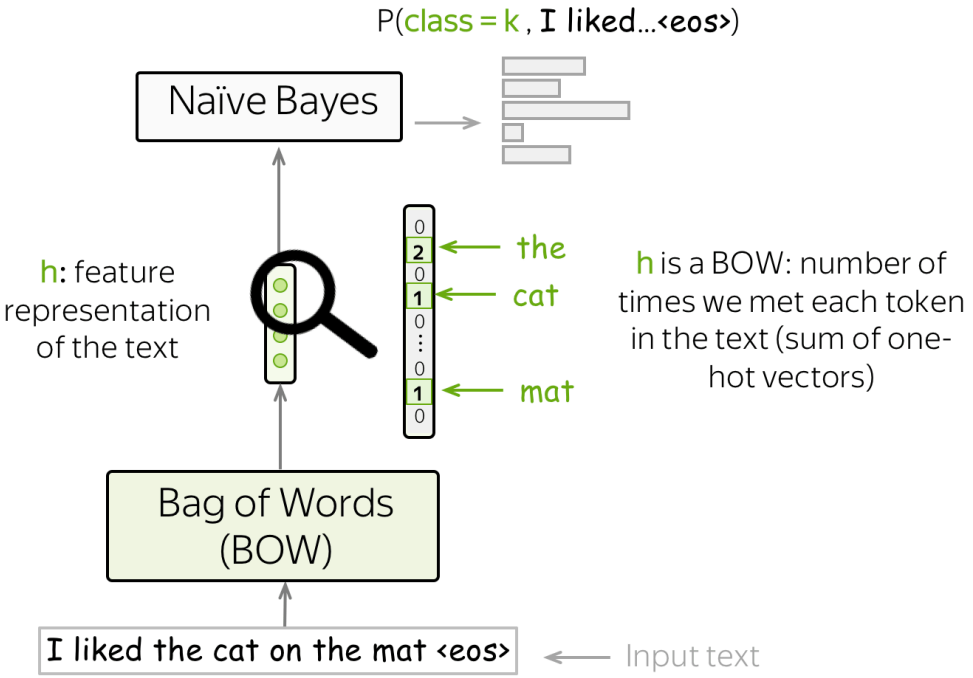
Since we care only about argmax, we can consider $\log P(x, y)$ instead of $P(x, y)$.

Important! Note that in practice, we will usually deal with log-probabilities and not probabilities.

View in the General Framework

Remember our general view on the classification task? We obtain feature representation of the input text using some method, then use this feature representation for classification.

In Naive Bayes, our features are words, and the feature representation is the Bag-of-Words (BOW) representation - a sum of one-hot representations of words. Indeed, to evaluate $P(x, y)$ we only need to number of times each token appeared in the text.



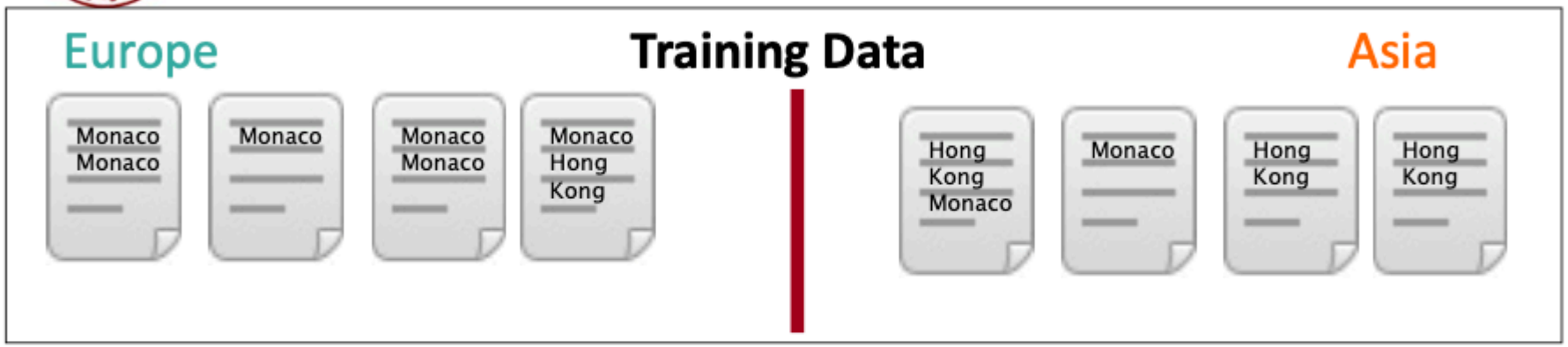
Feature Design

In the standard setting, we used words as features. However, you can use other types of features: URL, user id, etc.

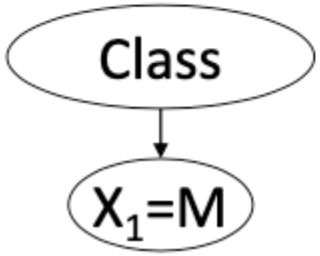
The problem of overcounting evidence from Naive Bayes

-based on Christopher Manning at Stanford

Text classification: Asia or Europe



NB Model



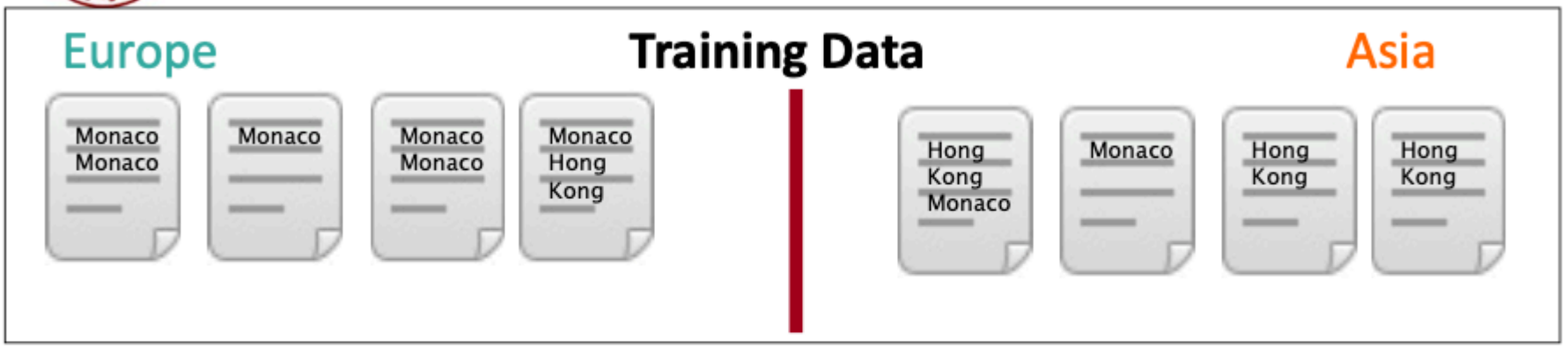
NB FACTORS:

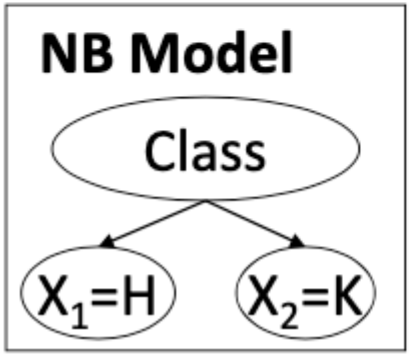
- P(A) = P(E) =
- P(M|A) =
- P(M|E) =

PREDICTIONS:

- P(A,M) =
- P(E,M) =
- P(A|M) =
- P(E|M) =

Text classification: Asia or Europe





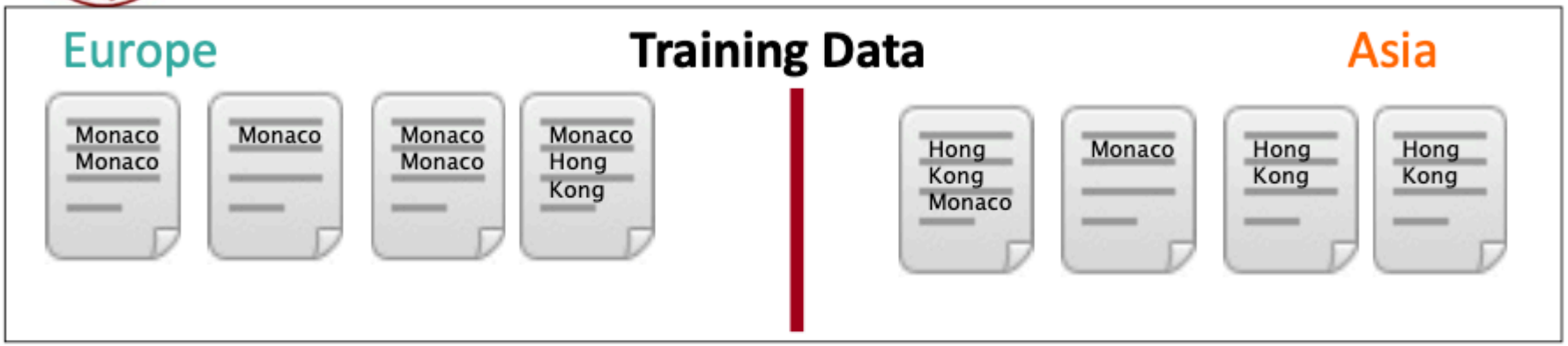
NB FACTORS:

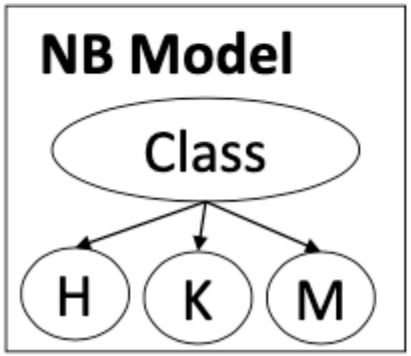
- P(A) = P(E) =
- P(H|A) = P(K|A) =
- P(H|E) = PK|E) =

PREDICTIONS:

- P(A,H,K) =
- P(E,H,K) =
- P(A|H,K) =
- P(E|H,K) =

Text classification: Asia or Europe





NB FACTORS:

- P(A) = P(E) =
- P(M|A) =
- P(M|E) =
- P(H|A) = P(K|A) =
- P(H|E) = PK|E) =

PREDICTIONS:

- P(A,H,K,M) =
- P(E,H,K,M) =
- P(A|H,K,M) =
- P(E|H,K,M) =

Naive Bayes vs. Maxent Models

- Naive Bayes models multi-count correlated evidence
  - Each feature is multiplied in, even when you have multiple features telling you the same thing
- Maximum Entropy models (pretty much) solve this problem
  - As we will see, this is done by weighting features so that model expectations match the observed (empirical) expectations

Maximum Entropy Classifier (aka Logistic Regression)

Differently from Naive Bayes, MaxEnt classifier is a discriminative model, i.e., we are interested in $P(y = k|x)$ and not in the joint distribution $p(x, y)$. Also, we will learn how to use features: this is in contrast to Naive Bayes, where we defined how to use the features ourselves.

Here we also have to define features manually, but we have more freedom: features do not have to be categorical (in Naive Bayes, they had to!). We can use the BOW representation or come up with something more interesting.

The general classification pipeline here is as follows:

- get $h = (f_1, f_2, \dots, f_n)$ - feature representation of the input text;
- take $w^{(i)} = (w_1^{(i)}, \dots, w_n^{(i)})$ - vectors with feature weights for each of the classes;
- for each class, weigh features, i.e. take the dot product of feature representation h with feature weights $w^{(k)}$:

$$w^{(k)}h = w_1^{(k)} \cdot f_1 + \cdots + w_n^{(k)} \cdot f_n, \quad k = 1, \ldots, K.$$

To get a bias term in the sum above, we define one of the features being 1 (e.g., $f_0 = 1$. Then

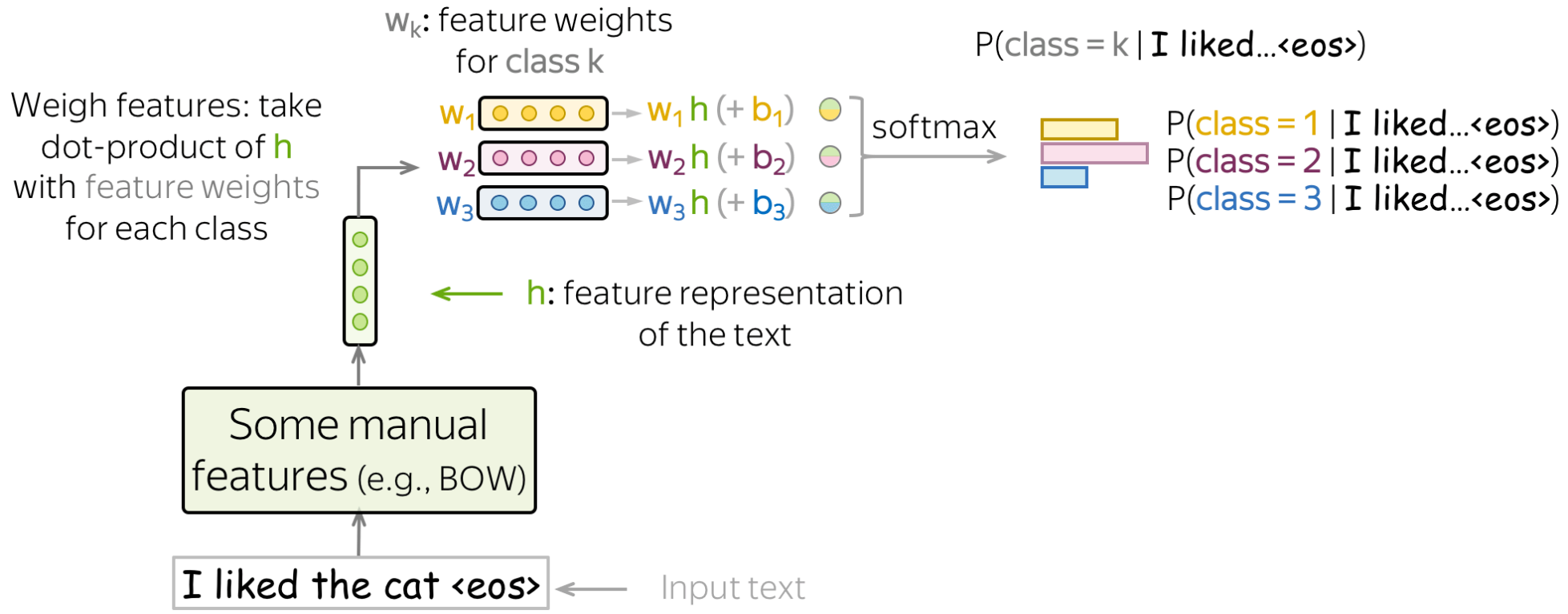
$$w^{(k)}h = w_0^{(k)} + w_1^{(k)} \cdot f_1 + \cdots + w_n^{(k)} \cdot f_n, \quad k = 1, \ldots, K.$$

- get class probabilities using softmax:

$$P(class = k|h) = \frac{\exp(w^{(k)}h)}{\sum_{i=1}^{K} \exp(w^{(i)}h)}.$$

Softmax normalizes the K values we got at the previous step to a probability distribution over output classes.

Look at the illustration below (classes are shown in different colors).



Training: Maximum Likelihood Estimate

Given training examples $x^1, \ldots, x^N$ with labels $y^1, \ldots, y^N$, $y^i \in \{1, \ldots, K\}$, we pick those weights $w^{(k)}, k = 1..K$ which maximize the probability of the training data:

$$w^* = \arg\max_w \sum_{i=1}^{N} \log P(y = y^i | x^i).$$

In other words, we choose parameters such that the data is more likely to appear. Therefore, this is called the **Maximum Likelihood Estimate (MLE) of the parameters.

To find the parameters maximizing the data log-likelihood, we use gradient ascent: gradually improve weights during multiple iterations over the data. At each step, we maximize the probability a model assigns to the correct class.

Equvalence to minimizing cross-entropy

Note that maximizing data log-likelihood is equivalent to minimizing cross entropy between the target probability distribution $p^* = (0, \ldots, 0, 1, 0, \ldots)$ (1 for the target label, 0 for the rest) and the predicted by the model distribution $p = (p_1, \ldots, p_K), p_i = p(i|x)$:

$$Loss(p^*, p) = -p^* \log(p) = -\sum_{i=1}^{K} p_i^* \log(p_i).$$

Since only one of p_i^* is non-zero (1 for the target label k, 0 for the rest), we will get $Loss(p^*, p) = -\log(p_k) = -\log(p(k|x))$.

Training example: **I liked the cat on the mat <eos>** Label: k

target

$$\log P(y = k|x) \rightarrow max$$

Maximizing log-likelihood of the correct class

$$\Updownarrow$$

$$-\log P(y = k|x) \rightarrow min$$

Minimizing negative log-likelihood of the correct class

$$\Updownarrow$$

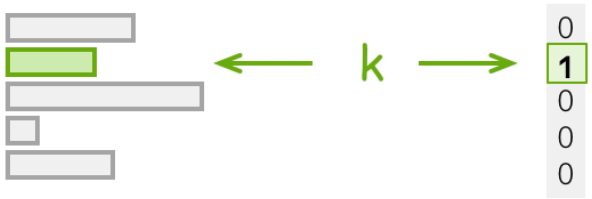
$$-\sum_{i=1}^{K} p_i^* \cdot \log P(y = i|x) \rightarrow min$$

Minimizing cross-entropy loss

$$(p_k^* = 1, p_i^* = 0, i \neq k)$$

Model prediction: P(class = i |**I liked…<eos>**)

Target: p^*



This equivalence is very important for you to understand: when talking about neural approaches, people usually say that they minimize the cross-entropy loss. Do not forget that this is the same as maximizing the data log-likelihood.

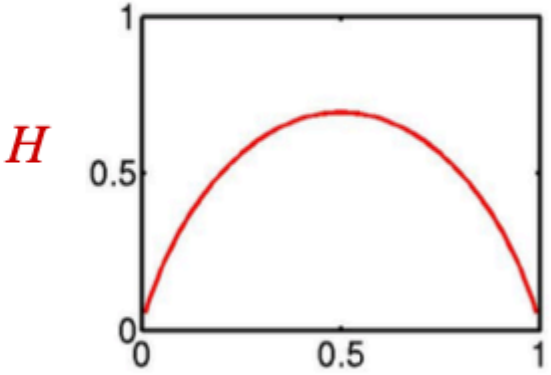
Maximum Entropy Roud Up

- based on Christopher Manning (Stanford)

(Maximum) Entropy

- Entropy: the uncertainty of a distribution.
- Quantifying uncertainty ("surprise"):
  - Event x
  - Probability p_x
  - "Surprise" $\log(1/p_x)$
- Entropy: expected surprise (over p):

$$\mathrm{H}(p) = E_p\left[\log_2 \frac{1}{p_x}\right] = -\sum_x p_x \log_2 p_x$$



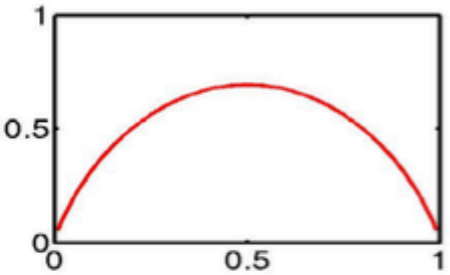
A coin-flip is most uncertain for a fair coin.

Maxent Examples I

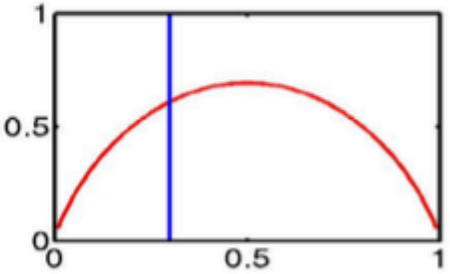
- What do we want from a distribution?
  - Minimize commitment = maximize entropy.
  - Resemble some reference distribution (data).
- Solution: maximize entropy H, subject to feature-based constraints:

$$E_p[f_i] = E_{\hat{p}}[f_i] \Leftrightarrow \sum_{x \in f_i} p_x = C_i$$

- Adding constraints (features):
  - Lowers maximum entropy
  - Raises maximum likelihood of data
  - Brings the distribution further from uniform
  - Brings the distribution closer to data

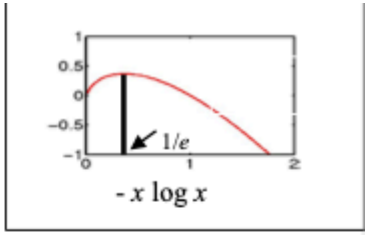


Unconstrained, max at 0.5

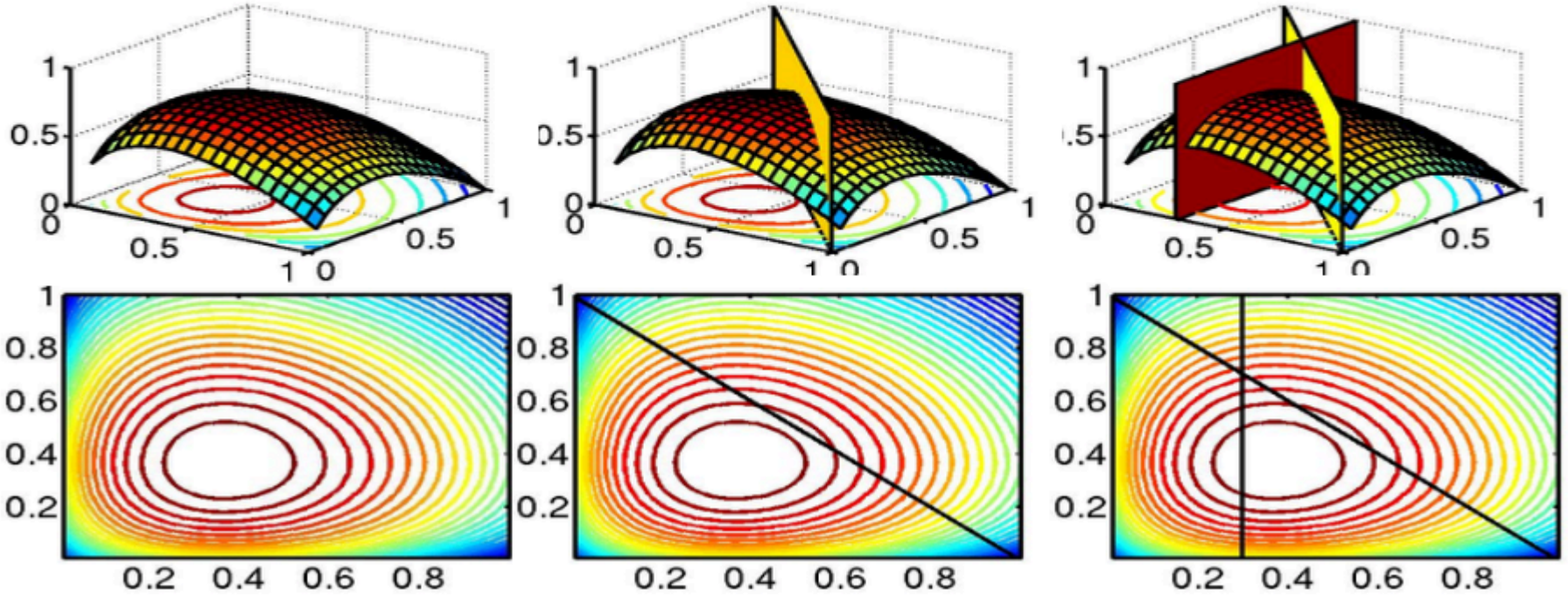


Constraint that $p_{\text{HEADS}} = 0.3$

Maxent Examples II



$H(p_H p_T,)$ $p_H + p_T = 1$ $p_H = 0.3$



Maxent Examples III

- Let's say we have the following event space:

NN	NNS	NNP	NNPS	VBZ	VBD

- … and the following empirical data:

3	5	11	13	3	1

- Maximize H:

$1/e$	$1/e$	$1/e$	$1/e$	$1/e$	$1/e$

- … want probabilities: E[NN,NNS,NNP,NNPS,VBZ,VBD] = 1

1/6	1/6	1/6	1/6	1/6	1/6

Maxent Examples IV

- Too uniform!
- N* are more common than V*, so we add the feature f_N = {NN, NNS, NNP, NNPS}, with E[f_N] =32/36

NN	NNS	NNP	NNPS	VBZ	VBD
8/36	8/36	8/36	8/36	2/36	2/36

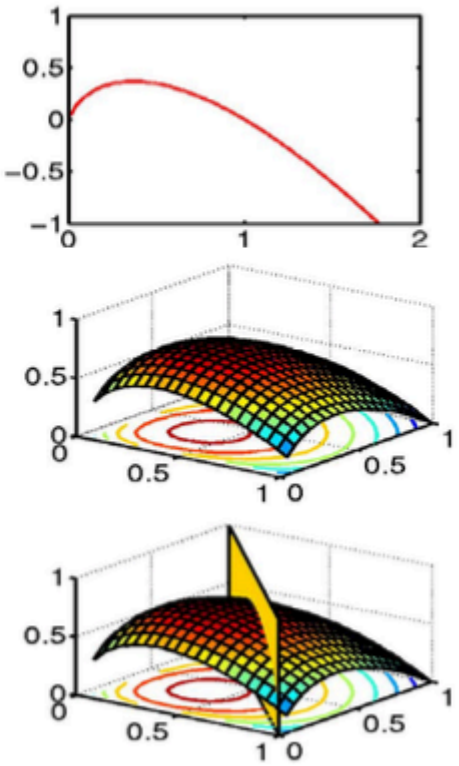
- … and proper nouns are more frequent than common nouns, so we add f_P = {NNP, NNPS}, with E[f_P] =24/36

4/36	4/36	12/36	12/36	2/36	2/36

- … we could keep refining the models, e.g., by adding a feature to distinguish singular vs. plural nouns, or verb types.

Convexity II

- Constrained $H(p) = -\sum x \log x$ is convex:
  - $-x \log x$ is convex
  - $-\sum x \log x$ is convex (sum of convex functions is convex).
  - The feasible region of constrained H is a linear subspace (which is convex)
  - The constrained entropy surface is therefore convex.
- The maximum likelihood exponential model (dual) formulation is also convex.



Feature Overlap

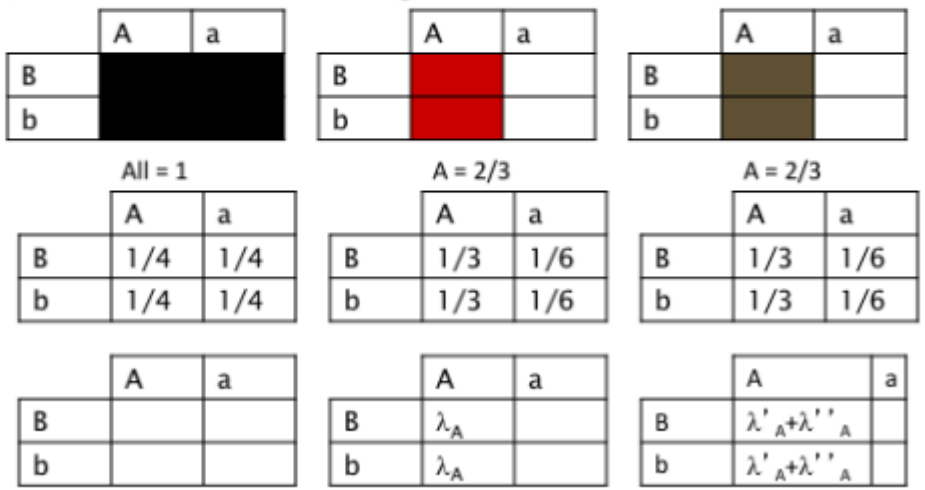
Christopher Manning

Feature Overlap

- Maxent models handle overlapping features well.
- Unlike a NB model, there is no double counting!

Empirical

	A	a
B	2	1
b	2	1



Naive Bayes vs Logistic Regression

Naïve Bayes:

- very simple
- very fast
- interpretable
- assumes that features are conditionally independent
- text representation: manually defined (and often too simple, e.g. BOW)

Logistic Regression:

- quite simple
- interpretable
- does **not** assume that features are conditionally independent
- not so fast (multiple iterations of gradient ascent)
- text representation: manually defined

Let's finalize this part by discussing the advantages and drawbacks of logistic regression and Naive Bayes.

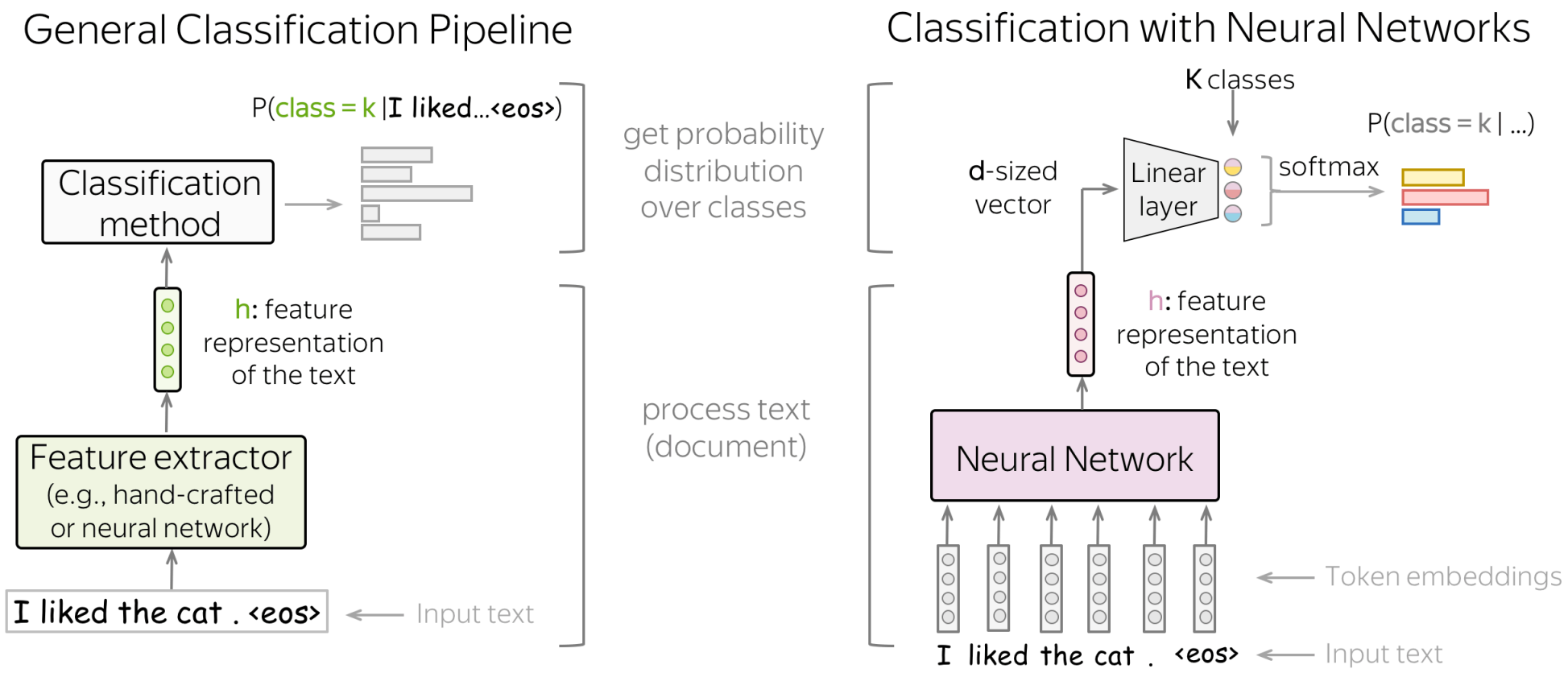
- simplicity
  Both methods are simple; Naive Bayes is the simplest one.
- interpretability
  Both methods are interpretable: you can look at the features which influenced the predictions most (in Naive Bayes - usually words, in logistic regression - whatever you defined).
- training speed
  Naive Bayes is very fast to train - it requires only one pass through the training data to evaluate the counts. For logistic regression, this is not the case: you have to go over the data many times until the gradient ascent converges.
- independence assumptions
  Naive Bayes is too "naive" - it assumed that features (words) are conditionally independent given class. Logistic regression does not make this assumption - we can hope it is better.
- text representation: manual
  Both methods use manually defined feature representation (in Naive Bayes, BOW is the standard choice, but you still choose this yourself). While

manually defined features are good for interpretability, they may be no so good for performance - you are likely to miss something which can be useful for the task.

Text Classification with Neural Networks

Instead of manually defined features, let a neural network to **learn useful features**.

The main idea of neural-network-based classification is that feature representation of the input text can be obtained using a neural network. In this setting, we feed the embeddings of the input tokens to a neural network, and this neural network gives us a vector representation of the input text. After that, this vector is used for classification.



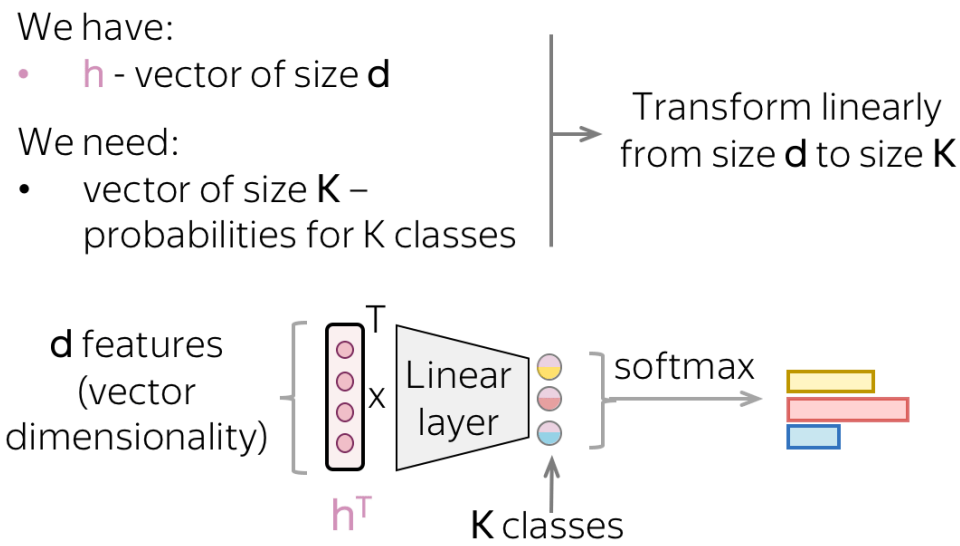
When dealing with neural networks, we can think about the classification part (i.e., how to get class probabilities from a vector representation of a text) in a very simple way.

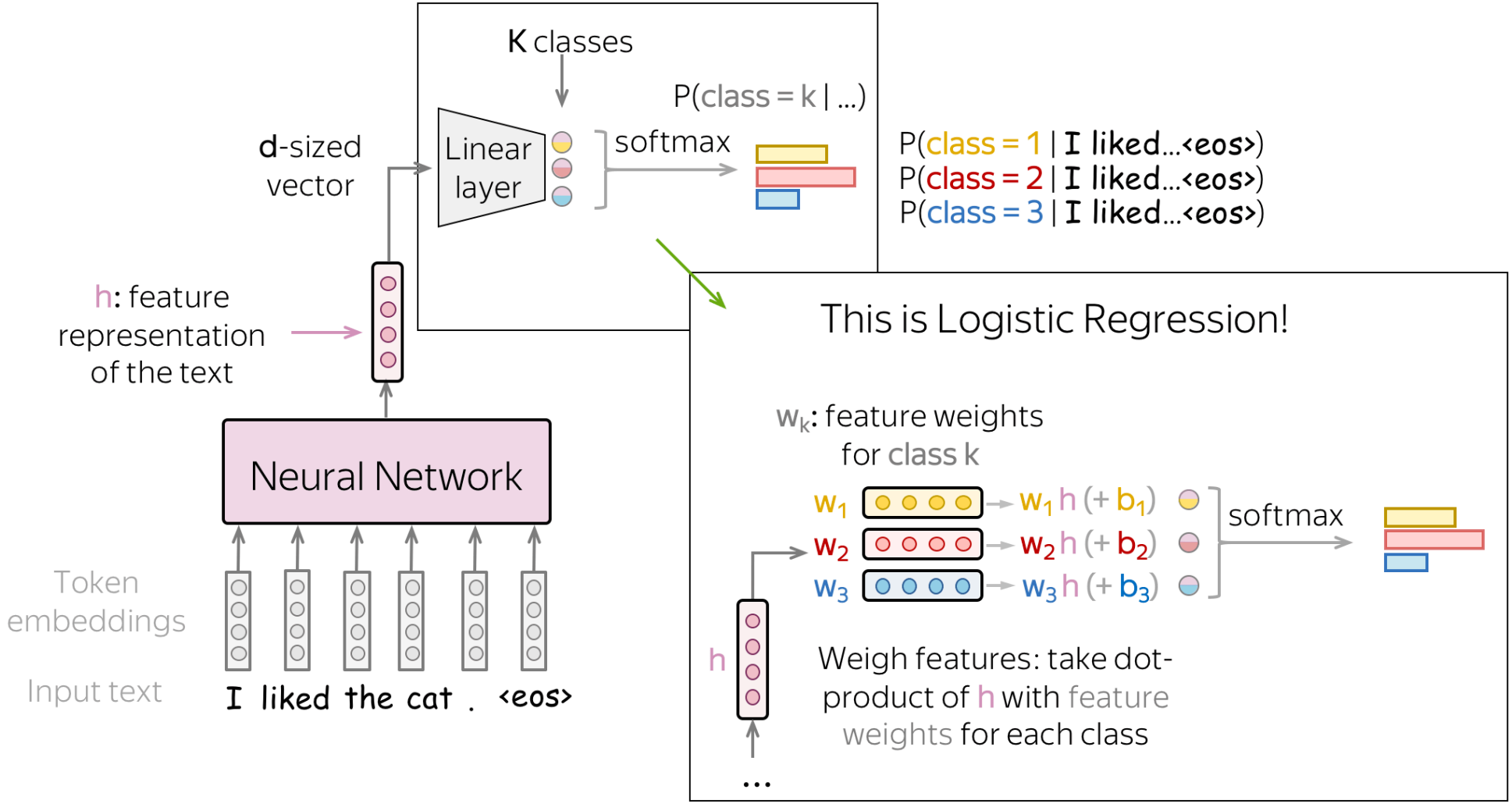
Vector representation of a text has some dimensionality d, but in the end, we need a vector of size K (probabilities for K classes). To get a K-sized vector from a d-sized, we can use a linear layer. Once we have a K-sized vector, all is left is to apply the softmax operation to convert the raw numbers

into class probabilities.

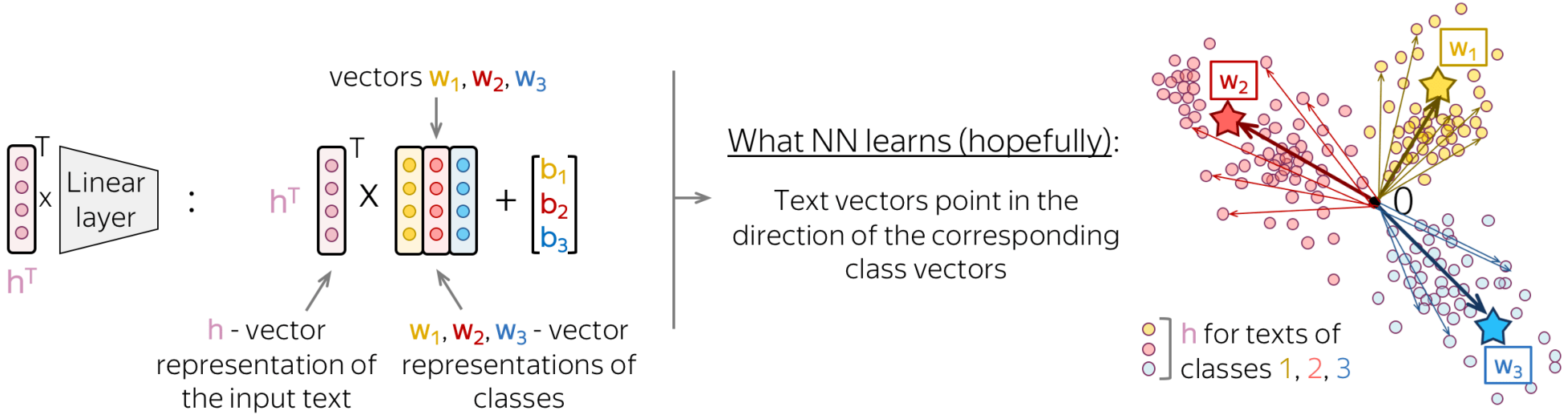
Classification Part: This is Logistic Regression!

Let us look closer to the neural network classifier. The way we use vector representation of the input text is exactly the same as we did with logistic regression: we weigh features according to feature weights for each class. The only difference from logistic regression is where the features come from: they are either defined manually (as we did before) or obtained by a neural network.





Intuition: Text Representation Points in the Direction of Class Representation



If we look at this final linear layer more closely, we will see that the columns of its matrix are vectors w_i. These vectors can be thought of as vector representations of classes. A good neural network will learn to represent input texts in such a way that text vectors will point in the direction of the corresponding class vectors.

Training and the Cross-Entropy Loss

Neural classifiers are trained to predict probability distributions over classes. Intuitively, at each step we maximize the probability a model assigns to the correct class.

The standard loss function is the cross-entropy loss. Cross-entropy loss for the target probability distribution $p^* = (0, \dots, 0, 1, 0, \dots)$ (1 for the target label, 0 for the rest) and the predicted by the model distribution $p = (p\_1, \dots, p\_K), p\_i = p(i|x)$:

$$Loss(p^*, p) = -p^* \log(p) = -\sum\_i = 1^K p\_i^* \log(p\_i).$$

Since only one of p_i^* is non-zero (1 for the target label k, 0 for the rest), we will get $Loss(p^*, p) = -\log(p\_k) = -\log(p(k|x))$. Look at the illustration for one training example.

Training example: I liked the cat on the mat <eos> Label: k

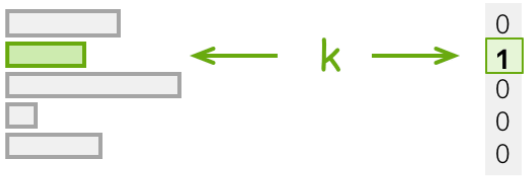
target

Model prediction:

P(class = i | I liked...<eos>)

Target:

p^*



Cross-entropy loss:

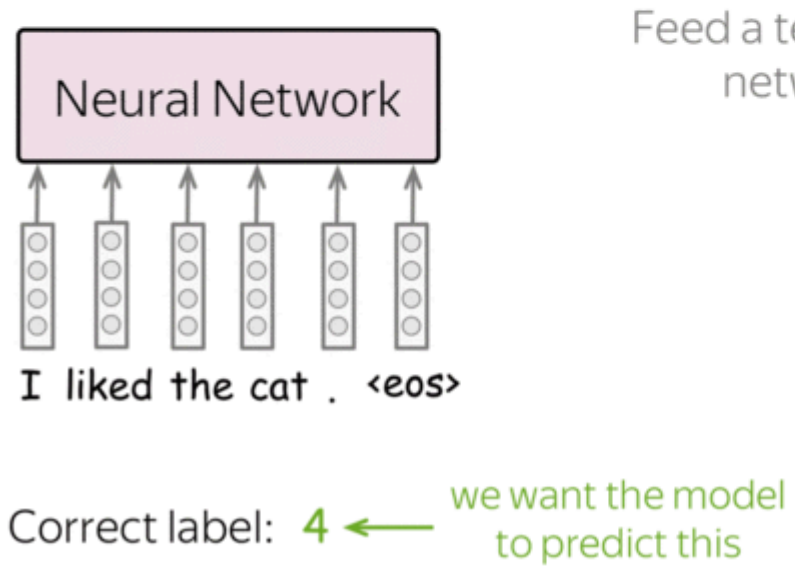
$$-\sum_{i=1}^{K} p_i^* \cdot \log P(y=i|x) \to min \qquad (p_k^* = 1, p_i^* = 0, i \neq k)$$

For one-hot targets, this is equivalent to

$$-\log P(y=k|x) \to min$$

In training, we gradually improve model weights during multiple iterations over the data: we iterate over training examples (or batches of examples) and make gradient updates. At each step, we maximize the probability a model assigns to the correct class. At the same time, we minimize sum of the probabilities of incorrect classes: since sum of all probabilities is constant, by increasing one probability we decrease sum of all the rest (Lena: Here I usually imagine a bunch of kittens eating from the same bowl: one kitten always eats at the expense of the others).

Look at the illustration of the training process.



Recap: This is equivalent to maximizing the data likelihood

Do not forget that when talking about MaxEnt classifier (logistic regression), we showed that minimizing cross-entropy is equivalent to maximizing the data likelihood. Therefore, here we are also trying to get the Maximum Likelihood Estimate (MLE) of model parameters.