Linear Models for Classification: Features & Weights

Nathan Schneider
(some slides borrowed from Chris Dyer)
ENLP | 3 February 2020

Outline

- **Words, probabilities → Features, weights**
- **Geometric view: decision boundary**

this lecture

- Perceptron
- Generative vs. Discriminative
- More discriminative models: Logistic regression/MaxEnt; SVM
- Loss functions, optimization
- Regularization; sparsity

next lecture

Word Sense Disambiguation (WSD)

- Given a word **in context**, predict which sense is being used.
  - Evaluated on corpora such as **SemCor**, which is fully annotated for WordNet synsets.

- For example: consider joint POS & WSD classification for 'interest', with 3 senses:
  - N:financial (*I repaid the loan with interest*)
  - N:nonfinancial (*I read the news with interest*)
  - V:nonfinancial (*Can I interest you in a dessert?*)

Beyond BoW

- Neighboring words are relevant to this decision.
- More generally, we can define **features** of the input that may help identify the correct class.
  - Individual words
  - Bigrams (pairs of consecutive words: *Wall Street*)
  - Capitalization (*interest* vs. *Interest* vs. *INTEREST*)
  - Metadata: document genre, author, …
- These can be used in naïve Bayes: "bag of features"
  - With overlapping features, independence assumption is *even more naïve*: $\mathrm{p}(y \mid \mathbf{x}) \propto \mathrm{p}(y) \cdots \mathrm{p}(\text{Wall} \mid y)\, \mathrm{p}(\text{Street} \mid y)\, \mathrm{p}(\text{Wall Street} \mid y)$

Choosing Features

- Supervision means that we don't have to pre-specify the precise relationship between each feature and the classification outcomes.

- But domain expertise helps in choosing which kinds of features to include in the model. (words, subword units, metadata, ...)
  - And sometimes, highly task-specific features are helpful.

- The decision about what features to include in a model is called **feature engineering**.
  - (There are some algorithmic techniques, such as *feature selection*, that can assist in this process.)
  - More features = more flexibility, but also more expensive to train, more opportunity for overfitting.

Feature Extraction

	$\phi(\mathbf{x})$
bias	1
capitalized?	0
#wordsBefore	6
#wordsAfter	3
relativeOffset	0.66
leftWord=about	1
leftWord=best	0
rightWord=rates	1
rightWord=in	0
Wall	1
Street	1
vets	1
best	0
in	0
Wall Street	1
Street vets	1
vets raise	1
. . .	

$\mathbf{x}$ = Wall Street vets raise concerns about interest rates , politics

bias feature (≈class prior): value of 1 for every **x** so the learned weight will reflect prevalence of the class

- Turns the input into a table of features with real values (often binary: 0 or 1).
- In practice: define feature templates like "leftWord=•" from which specific features are instantiated

Feature Extraction

	$\phi(\mathbf{x})$
bias	1
capitalized?	0
#wordsBefore	6
#wordsAfter	3
relativeOffset	0.66
leftWord=about	1
leftWord=best	0
rightWord=rates	1
rightWord=in	0
Wall	1
Street	1
vets	1
best	0
in	0
Wall Street	1
Street vets	1
vets raise	1

. . .

$\mathbf{x}$ = Wall Street vets raise concerns about interest rates , politics

spelling feature

- Turns the input into a table of features with real values (often binary: 0 or 1).
- In practice: define feature templates like "leftWord=•" from which specific features are instantiated

Feature Extraction

x = Wall Street vets raise concerns about interest rates , politics

	$\phi(\mathbf{x})$
bias	1
capitalized?	0
#wordsBefore	6
#wordsAfter	3
relativeOffset	0.66
leftWord=about	1
leftWord=best	0
rightWord=rates	1
rightWord=in	0
Wall	1
Street	1
vets	1
best	0
in	0
Wall Street	1
Street vets	1
vets raise	1
...	

token positional features

- Turns the input into a table of features with real values (often binary: 0 or 1).
- In practice: define feature templates like "leftWord=•" from which specific features are instantiated

Feature Extraction

$\mathbf{x}$ = Wall Street vets raise concerns about interest rates , politics

	$\phi(\mathbf{x})$
bias	1
capitalized?	0
#wordsBefore	6
#wordsAfter	3
relativeOffset	0.66
leftWord=about	1
leftWord=best	0
rightWord=rates	1
rightWord=in	0
Wall	1
Street	1
vets	1
best	0
in	0
Wall Street	1
Street vets	1
vets raise	1
...	

immediately neighboring words

- Turns the input into a table of features with real values (often binary: 0 or 1).
- In practice: define feature templates like "leftWord=•" from which specific features are instantiated

Feature Extraction

x = Wall Street vets raise concerns about interest rates , politics

	$\phi(\mathbf{x})$
bias	1
capitalized?	0
#wordsBefore	6
#wordsAfter	3
relativeOffset	0.66
leftWord=about	1
leftWord=best	0
rightWord=rates	1
rightWord=in	0
Wall	1
Street	1
vets	1
best	0
in	0
Wall Street	1
Street vets	1
vets raise	1
...	

unigrams

- Turns the input into a table of features with real values (often binary: 0 or 1).
- In practice: define feature templates like “leftWord=•” from which specific features are instantiated

Feature Extraction

$\mathbf{x}$ = Wall Street vets raise concerns about interest rates , politics

	$\phi(\mathbf{x})$
bias	1
capitalized?	0
#wordsBefore	6
#wordsAfter	3
relativeOffset	0.66
leftWord=about	1
leftWord=best	0
rightWord=rates	1
rightWord=in	0
Wall	1
Street	1
vets	1
best	0
in	0
Wall Street	1
Street vets	1
vets raise	1
...	

bigrams

- Turns the input into a table of features with real values (often binary: 0 or 1).
- In practice: define feature templates like "leftWord=•" from which specific features are instantiated

Feature Extraction

x = Wall Street vets raise concerns about interest rates , politics

x′ = Pet 's best interest in mind , but vets must follow law

	$\phi(\mathbf{x})$	$\phi(\mathbf{x}')$
bias	1	1
capitalized?	0	0
#wordsBefore	6	3
#wordsAfter	3	8
relativeOffset	0.66	0.27
leftWord=about	1	0
leftWord=best	0	1
rightWord=rates	1	0
rightWord=in	0	1
Wall	1	0
Street	1	0
vets	1	1
best	0	1
in	0	1
Wall Street	1	0
Street vets	1	0
vets raise	1	0

...

- Turns the input into a table of features with real values (often binary: 0 or 1).
- In practice: define feature templates like "leftWord=•" from which specific features are instantiated

Linear Model

- For each input $\mathbf{x}$ (e.g., a document or word token), let $\boldsymbol{\phi}(\mathbf{x})$ be a function that extracts a vector of its features.
    - Features may be binary (e.g., `capitalized?`) or real-valued (e.g., `#word=debt`).

- Each feature receives a real-valued **weight** parameter w . Each candidate label y' is scored for the token by summing the weights for the active features:

$$\mathbf{w}_{y'}{}^{\top}\boldsymbol{\phi}(\mathbf{x})$$
$$= \Sigma_j \, w_{y',j} \cdot \phi_j(\mathbf{x})$$

- For binary classification, equivalent to: $\mathrm{sign}(\mathbf{w}^{\top}\boldsymbol{\phi}(\mathbf{x}))$ — $+1$ or -1

	$\phi(\mathbf{x})$	w	$\phi(\mathbf{x}')$
bias	1	−3.00	1
capitalized?	0	.22	0
#wordsBefore	6	−.01	3
#wordsAfter	3	.01	8
relativeOffset	0.6	1.00	0.2
leftWord=about	1	.00	0
leftWord=best	0	−2.00	1
rightWord=rates	1	5.00	0
rightWord=in	0	−1.00	1
Wall	1	1.00	0
Street	1	−1.00	0
vets	1	−.05	1
best	0	−1.00	1
in	0	−.01	1
Wall Street	1	4.00	0
Street vets	1	.00	0
vets raise	1	.00	0

. . .

$\mathbf{x}$ = Wall Street vets raise concerns about interest rates , politics

$\mathbf{x}'$ = Pet 's best interest in mind , but vets must follow law

- Weights are learned from data
- For the moment, assume binary classification: financial or nonfinancial
  - More positive weights more indicative of financial.
  - $\mathbf{w}^{\mathsf{T}}\phi(\mathbf{x}) = 6.59$, $\mathbf{w}^{\mathsf{T}}\phi(\mathbf{x}') = -6.74$

More then 2 classes

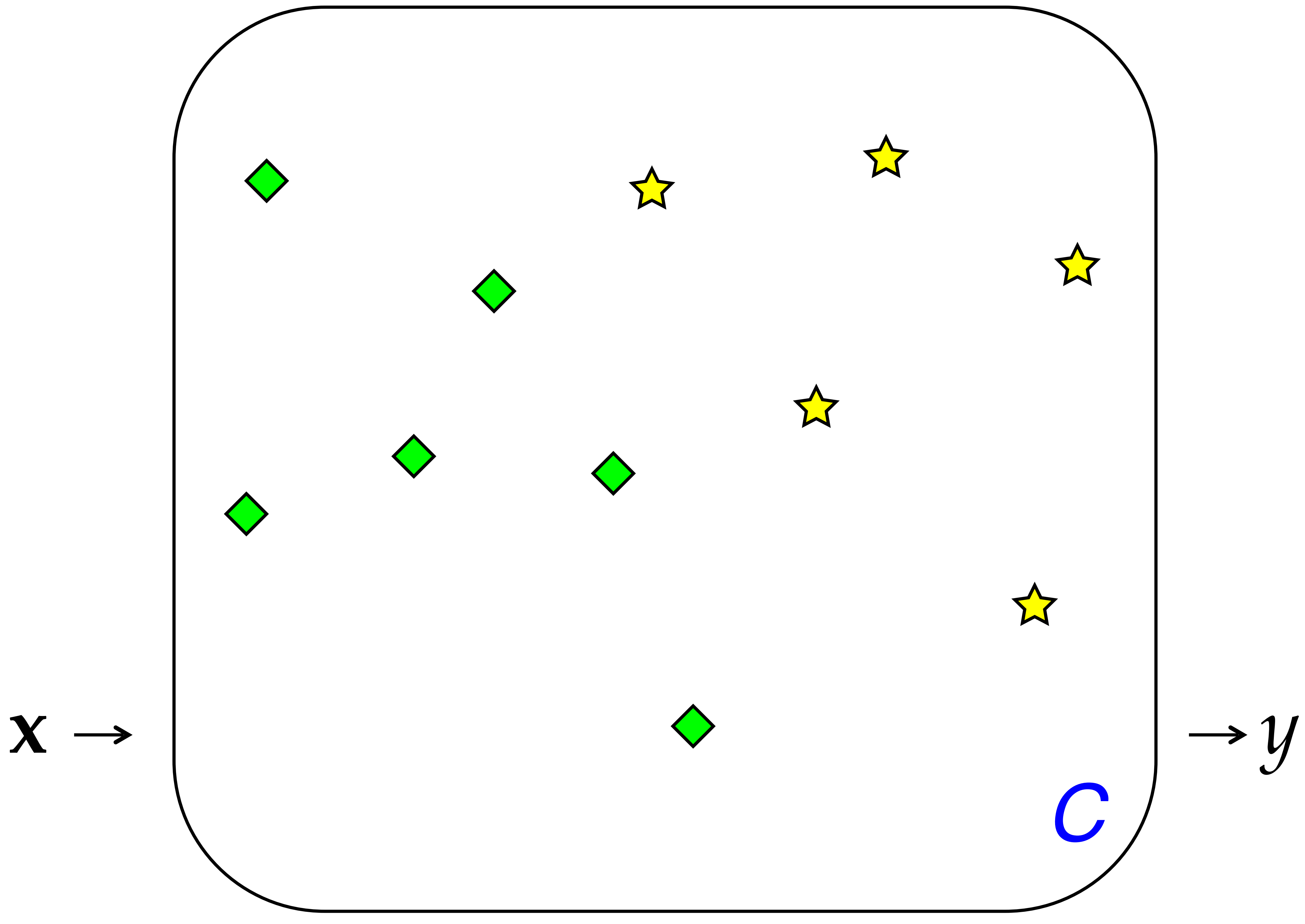
- Simply keep a separate weight vector for each class: $\mathbf{w}_y$

- The class whose weight vector gives the highest score wins!

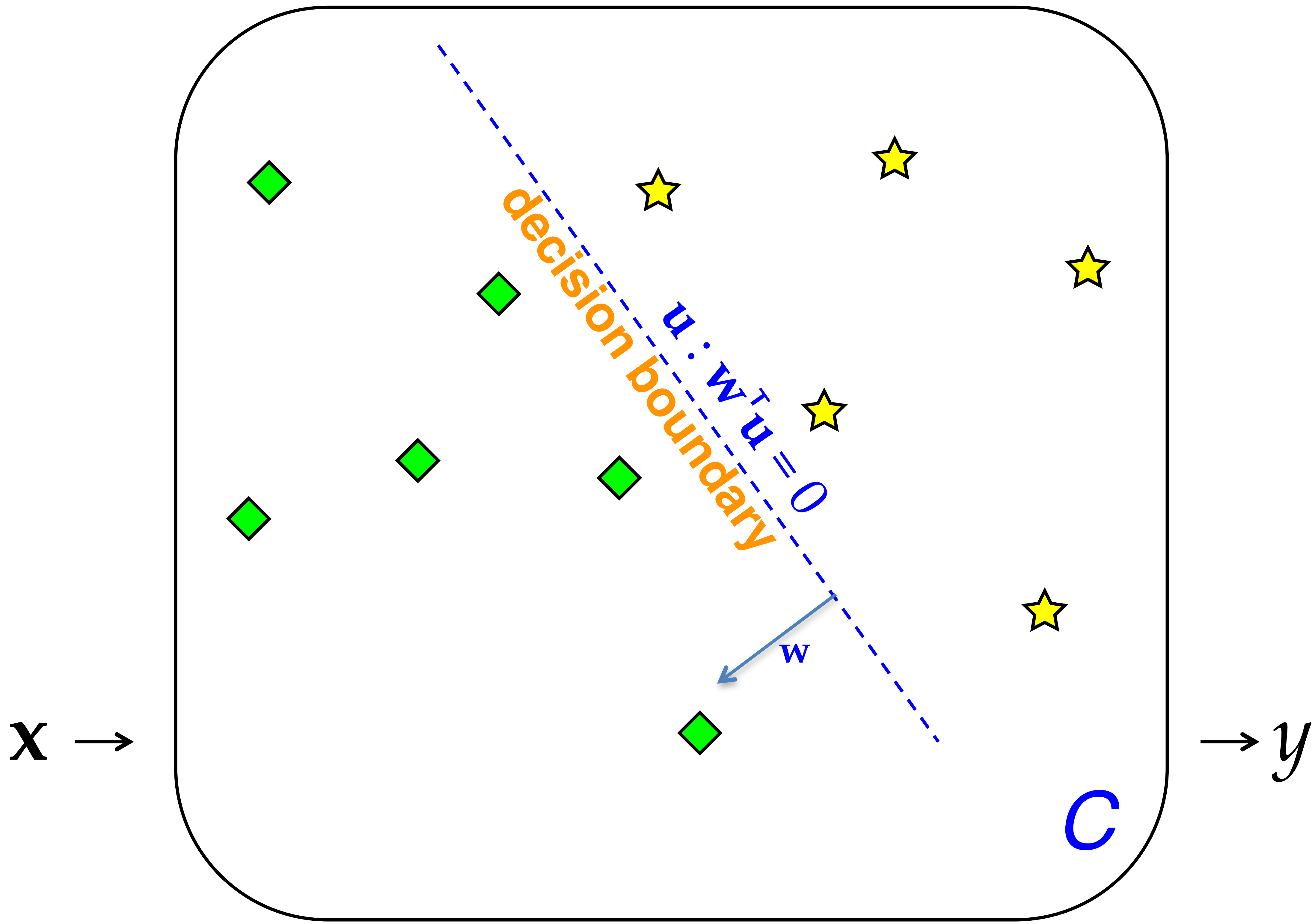
Learning the weights

- Weights depend on the choice of model and learning algorithm.
- Naïve Bayes fits into this framework, under the following estimation procedure for $\mathbf{w}$:
  - $w_{\text{bias}} = \log \mathrm{p}(y)$
  - $\forall$ features f: $w_f = \log \mathrm{p}(f \mid y)$
  - $\Sigma_j\, w_j \cdot \phi_j(\mathbf{x}) = w_{\text{bias}} + \Sigma_{f \in \phi(\mathbf{x})}\, w_f$
    $= \log \mathrm{p}(y) + \Sigma_{f \in \phi(\mathbf{x})} \log \mathrm{p}(f \mid y)$
    $= \log\, (\mathrm{p}(y) \cdot \Pi_{f \in \phi(\mathbf{x})}\, \mathrm{p}(f \mid y))$
- However, the naïve independence assumption—that all features are conditionally independent given the class—can be harmful.
  - Could the weights shown on the previous slide be naïve Bayes estimates?
    - No, because some are positive (thus not log-probabilities). Other kinds of learning procedures can give arbitrary real-valued weights.
    - If using log probabilities as weights, then the classification threshold should be equivalent to probability of .5, i.e. **log .5**.

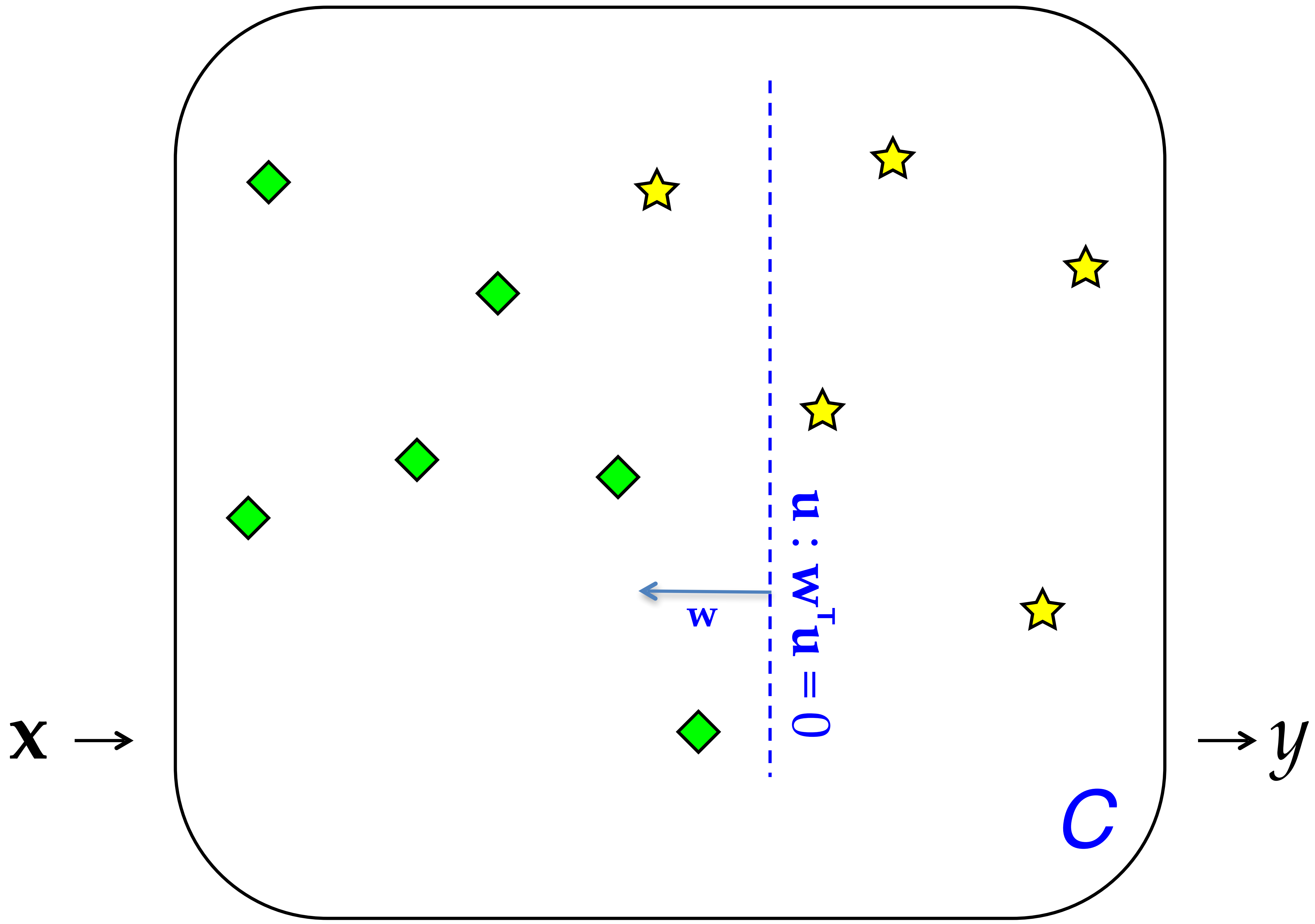
Linear Classifiers: Geometric View



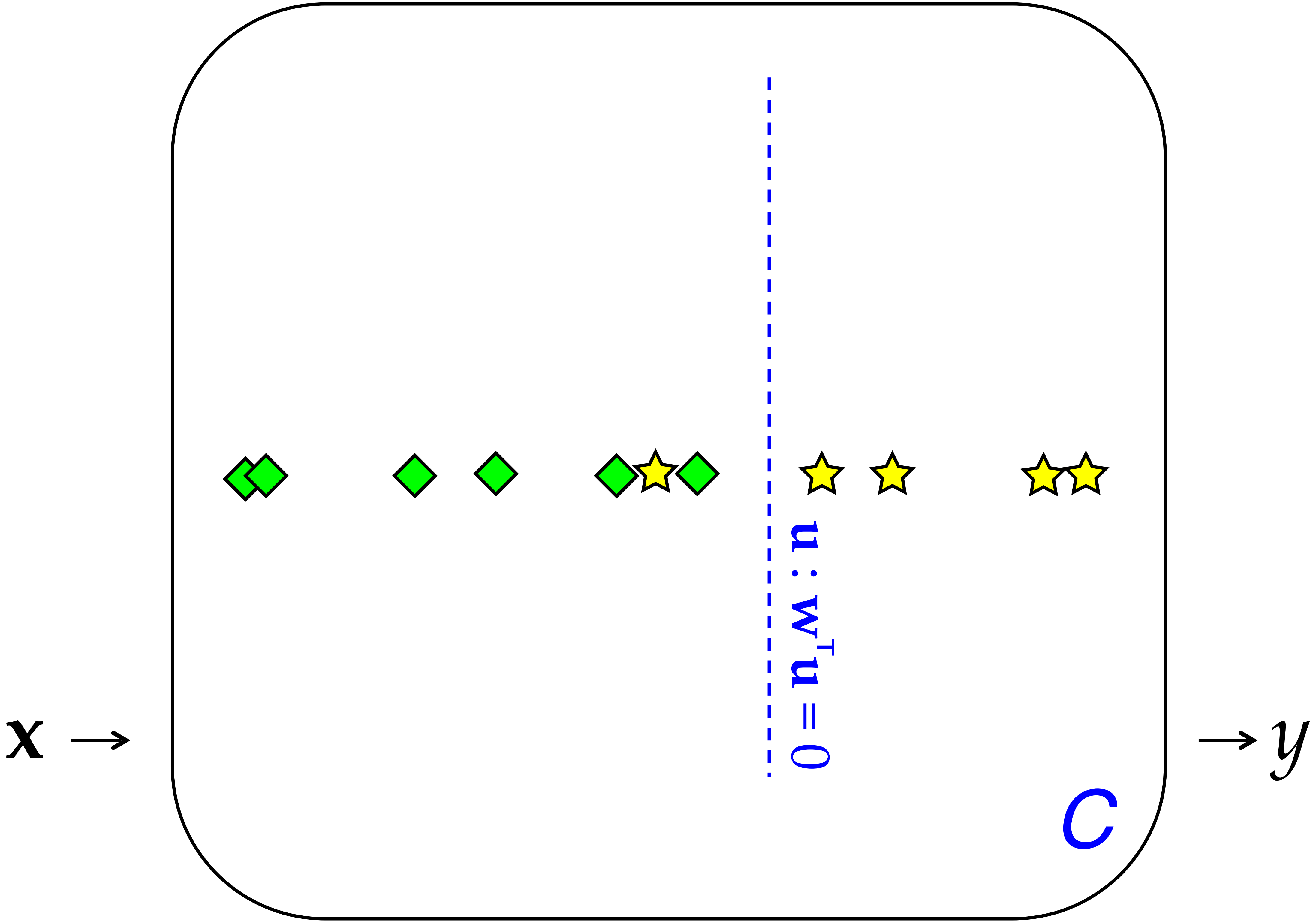
Linear Classifiers: Geometric View



Linear Classifiers: Geometric View



Linear Classifiers: Geometric View



Linear Classifiers (> 2 Classes)

$\mathbf{x} \rightarrow$

return

$\arg\max_y \mathbf{w}_y^\top \boldsymbol{\Phi}(\mathbf{x})$

C

$\rightarrow y$

The term "feature"

- The term "feature" is overloaded in NLP/ML. Here are three different concepts:
  - **Linguistic feature:** in some formalisms, a symbolic property that applies to a unit to categorize it, e.g. [−voice] for a sound in phonology or [+past] for a verb in morphology.
  - **Percept** (or **input feature**)**:** captures some aspect of an input $\mathbf{x}$; binary- or real-valued. *[The term "percept" is nonstandard but I think it is useful!]*

    ends in *-ing*
  - **Parameter** (or **model feature**): an association between some percept and an output class (or structure) y for which a real-valued weight or score is learned.

    ends in *-ing* **∧** y=VERB

Linear Models for Classification: Discriminative Learning (Perceptron, SVMs, MaxEnt)

Nathan Schneider
(some slides borrowed from Chris Dyer)
ENLP | 5 February 2020

Outline

- Words, probabilities → Features, weights

previous lecture

- Geometric view: decision boundary

this lecture

- Perceptron
- Generative vs. Discriminative
- More discriminative models: Logistic regression/MaxEnt; SVM
- Loss functions, optimization
- Regularization; sparsity

Perceptron Learner

$\mathbf{X} \rightarrow$

$\mathbf{Y} \rightarrow$

$\mathbf{w} \leftarrow \mathbf{0}$

for i = 1 … I:

  for t = 1 … T:

    select $(\mathbf{x}, y)_t$

    # run current classifier

    $\hat{y} \leftarrow \arg\max_{y'} \mathbf{w}_{y'}^{\mathsf{T}} \boldsymbol{\Phi}(\mathbf{x})$

    if $\hat{y} \neq y$ then *# mistake*

      $\mathbf{w}_y \leftarrow \mathbf{w}_y + \boldsymbol{\Phi}(\mathbf{x})$

      $\mathbf{w}_{\hat{y}} \leftarrow \mathbf{w}_{\hat{y}} - \boldsymbol{\Phi}(\mathbf{x})$

return $\mathbf{w}$

L

$\rightarrow \mathbf{w}$

(assumes all classes have the same **percepts**)

Perceptron Learner

$\mathbf{X} \rightarrow$

$\mathbf{Y} \rightarrow$

$\mathbf{w} \leftarrow \mathbf{0}$

for i = 1 … I:

  for t = 1 … T:

    select $(\mathbf{x}, y)_t$

    # run current classifier

    $\hat{y} \leftarrow$ C $\leftarrow \mathbf{x}$

    decoding is a subroutine of learning

    if $\hat{y} \neq y$ then *# mistake*

      $\mathbf{w}_y \leftarrow \mathbf{w}_y + \mathbf{\Phi}(\mathbf{x})$

      $\mathbf{w}_{\hat{y}} \leftarrow \mathbf{w}_{\hat{y}} - \mathbf{\Phi}(\mathbf{x})$

return $\mathbf{w}$

L

$\rightarrow$ $\mathbf{w}$

(assumes all classes have the same **percepts**)

Perceptron Learner

for **binary classification**

single weight vector such that

>0 → + class, <0 → – class

$\mathbf{X} \rightarrow$

$\mathbf{Y} \rightarrow$

$\mathbf{w} \leftarrow \mathbf{0}$

for i = 1 … I:

for t = 1 … T:

select $(\mathbf{x}, y)_t$

 # run current classifier

$\hat{y} \leftarrow \text{sign}(\mathbf{w}^\top \mathbf{\Phi}(\mathbf{x}))$

if $\hat{y} \neq y$ then *# mistake*

$\mathbf{w} \leftarrow \mathbf{w} + \text{sign}(y) \cdot \mathbf{\Phi}(\mathbf{x})$

return $\mathbf{w}$

L

$\rightarrow \mathbf{w}$

(assumes all classes have the same **percepts**)

Perceptron Learner

$\mathbf{X} \rightarrow$

$\mathbf{Y} \rightarrow$

$\mathbf{w} \leftarrow \mathbf{0}$

for i = 1 … I:

  for t = 1 … T:

    select $(\mathbf{x}, y)_t$

    # run current classifier

    $\hat{y} \leftarrow \arg\max_{y'} \mathbf{w}^{\top} \mathbf{\Phi}(\mathbf{x}, y')$

if different classes have different **percepts**

    if $\hat{y} \neq y$ then *# mistake*

      $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{\Phi}(\mathbf{x}, y) - \mathbf{\Phi}(\mathbf{x}, \hat{y})$

return $\mathbf{w}$

L

$\rightarrow \mathbf{w}$

work through example on the board

x1 = "I thought it was great" y1 = +

x2 = "not so great" y2 = −

x3 = "good but not great" y3 = +

Perceptron Learner

- **The perceptron doesn't estimate probabilities.** It just adjusts weights up or down until they classify the training data correctly.
  - No assumptions of feature independence necessary! ⇒ Better accuracy than NB
- The perceptron is an example of an **online** learning algorithm because it potentially updates its parameters (weights) with each training datapoint.
- Classification, a.k.a. **decoding**, is called with the latest weight vector. Mistakes lead to weight updates.
- One hyperparameter: I, the number of iterations (passes through the training data).
- Often desirable to make several passes over the training data. The number can be tuned. Under certain assumptions, it can be proven that the learner will converge.

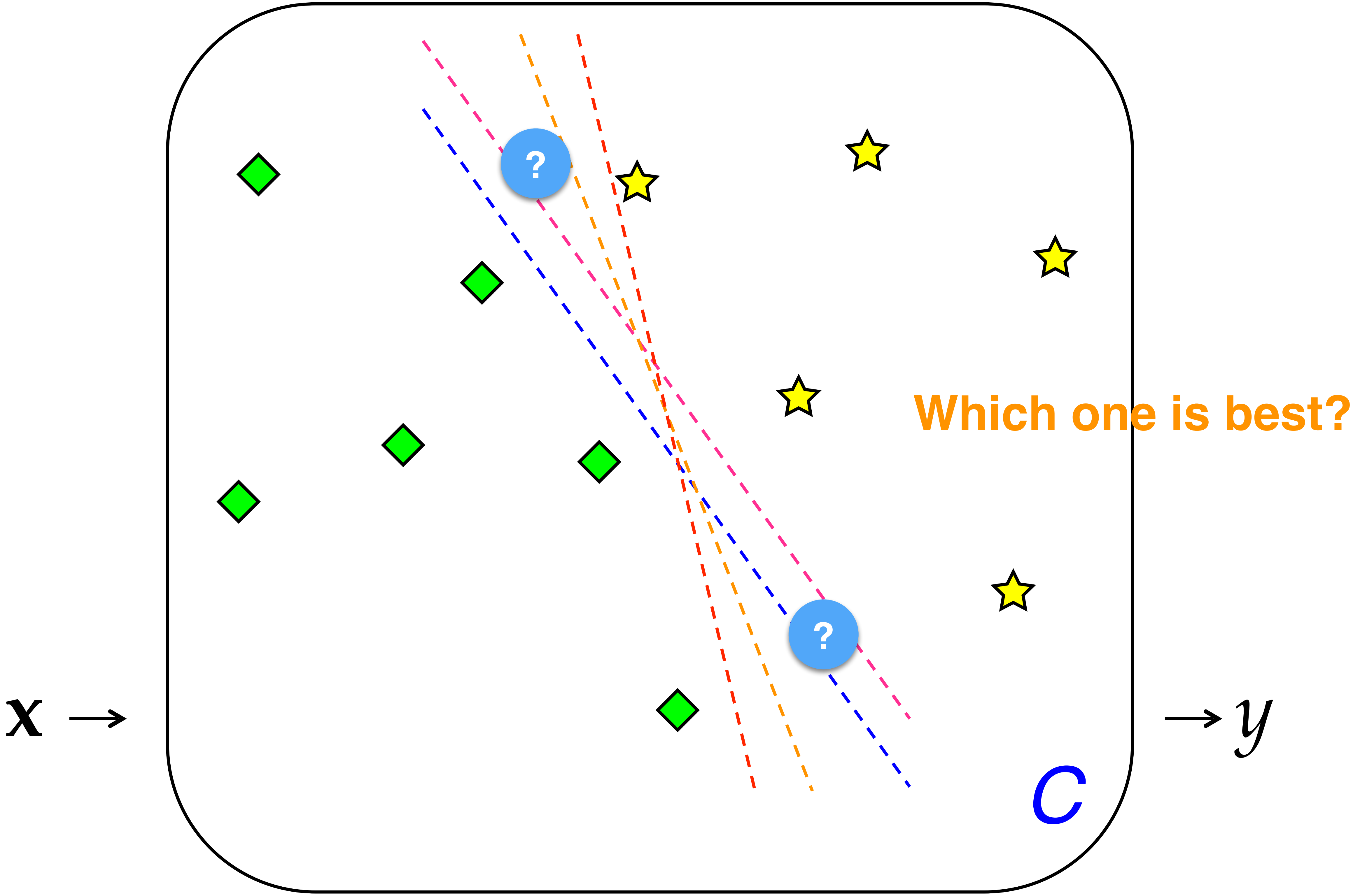
Perceptron: Avoiding overfitting

- Like any learning algorithm, the perceptron risks overfitting the training data. Two main techniques to improve generalization:
  - **Averaging:** Keep a copy of each weight vector as it changes, then average all of them to produce the final weight vector. Daumé chapter has a trick to make this efficient with large numbers of features.
  - **Early stopping:** Tune I by checking held-out accuracy on dev data (or cross-val on train data) after each iteration. If accuracy has ceased to improve, stop training and use the model from iteration $I-1$.

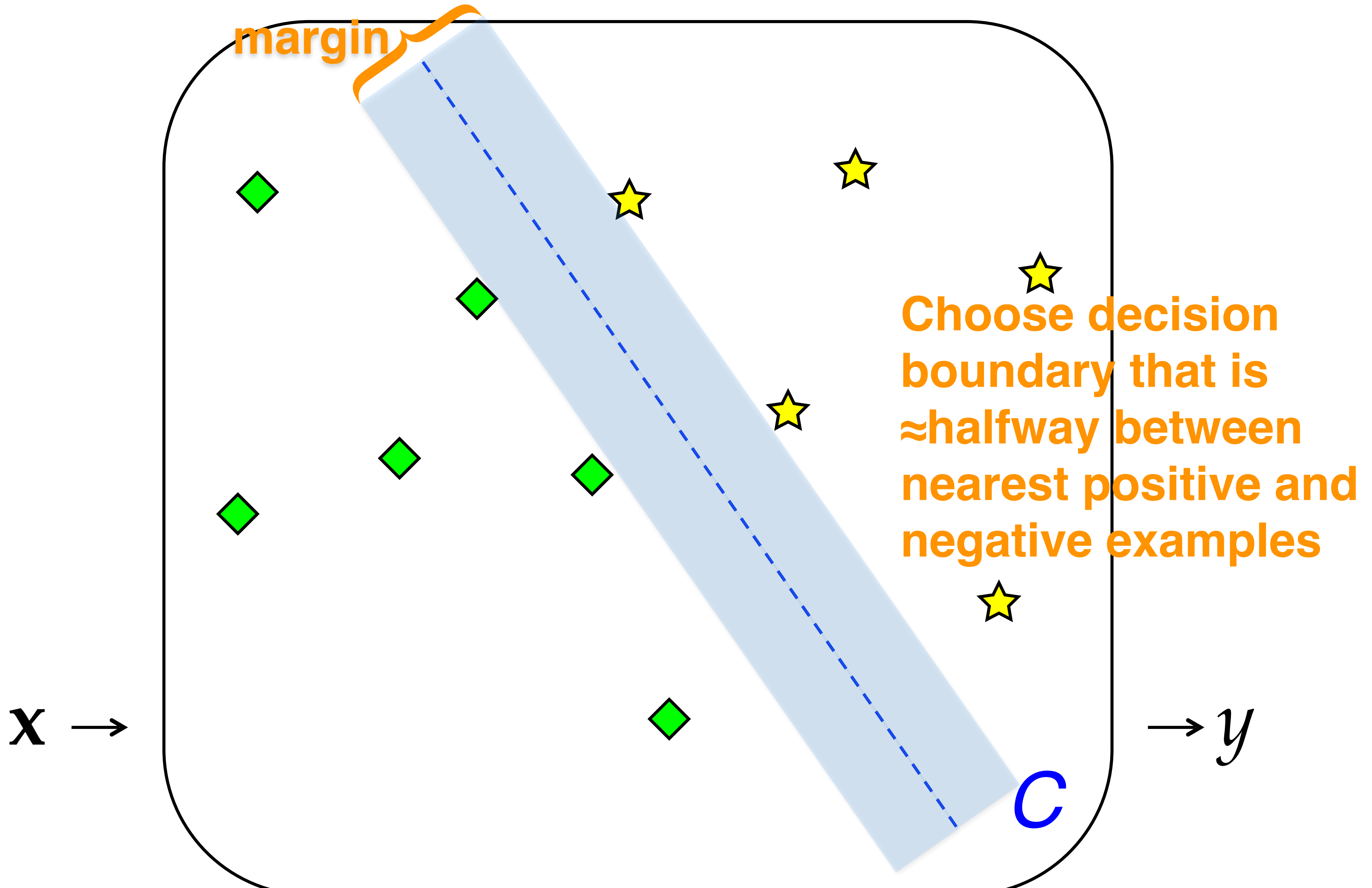
Generative vs. Discriminative

- Naïve Bayes allows us to classify via the **joint probability** of $\mathbf{x}$ and y:
  - $p(y \mid \mathbf{x}) \propto p(y) \prod_{w \in \mathbf{x}} p(w \mid y)$
    $= p(y)\, p(\mathbf{x} \mid y)$ (per the independence assumptions of the model)
    $= p(y, \mathbf{x})$ (chain rule)
  - This means the model accounts for BOTH **x** and *y*. From the joint distribution p(**x**,*y*) it is possible to compute p(**x**) as well as p(*y*), p(**x** | *y*), and p(*y* | **x**).
- NB is called a **generative** model because it assigns probability to linguistic objects (**x**). It could be used to generate “likely” language corresponding to some *y*. (Subject to its naïve modeling assumptions!)
  - (Not to be confused with the “generative” school of linguistics.)
- Some other linear models, including the perceptron, are **discriminative**: they are trained directly to classify given **x**, and cannot be used to estimate the probability of **x** or generate **x** | *y*.

Many possible decision boundaries



Max-Margin Methods (e.g., SVM)



Max-Margin Methods

- **Support Vector Machine (SVM)**: most popular max-margin variant
- Closely related to the perceptron; can be optimized (learned) with a slight tweak to the perceptron algorithm.
- Like perceptron, discriminative, non-probabilistic.

Maximum Entropy (MaxEnt) a.k.a. (Multinomial) Logistic Regression

- What if we want a discriminative classifier with **probabilities**?
  - E.g., need confidence of prediction, or want the full distribution over possible classes
- Wrap the linear score computation ($\mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y')$) in the **softmax** function:
  - score can be negative; exp(score) is always positive

    $$\log p(y \mid \mathbf{x}) = \log \frac{\exp(\mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y))}{\Sigma_{y'} \exp(\mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y'))} = \mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y) - \log \Sigma_{y'} \exp(\mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y'))$$

    Denominator = normalization (makes probabilities sum to 1).
    Sum over all classes ⇒ same for all numerators ⇒ can be ignored at classification time.
  - **Binary case:**

    $$\log p(y{=}1 \mid \mathbf{x}) = \log \frac{\exp(\mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y{=}1))}{\exp(\mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y{=}1)) + \exp(\mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y{=}0))}$$

    $$= \log \frac{\exp(\mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y{=}1))}{\exp(\mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y{=}1)) + 1}$$ (fixing $\mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y{=}0) = 0$)
- MaxEnt classifiers are a special case of **MaxEnt** a.k.a. **log-linear models**.
  - Why the term "Maximum Entropy"? See Smith *Linguistic Structure Prediction*, appendix C.

Objectives

- For all linear models, the **classification rule** or **decoding objective** is: $y \leftarrow \arg\max_{y'} \mathbf{w}^\intercal \mathbf{\Phi}(\mathbf{x}, y')$
  - Objective function = function for which we want to find the optimum (in this case, the max)
- There is also a **learning objective** for which we want to find the optimal **parameters**. Mathematically, NB, MaxEnt, SVM, and perceptron all optimize different learning objectives.
  - When the learning objective is formulated as a **minimization** problem, it's called a **loss** function.
  - A loss function scores the "badness" of the training data under any possible set of parameters. Learning = choosing the parameters that minimize the badness.

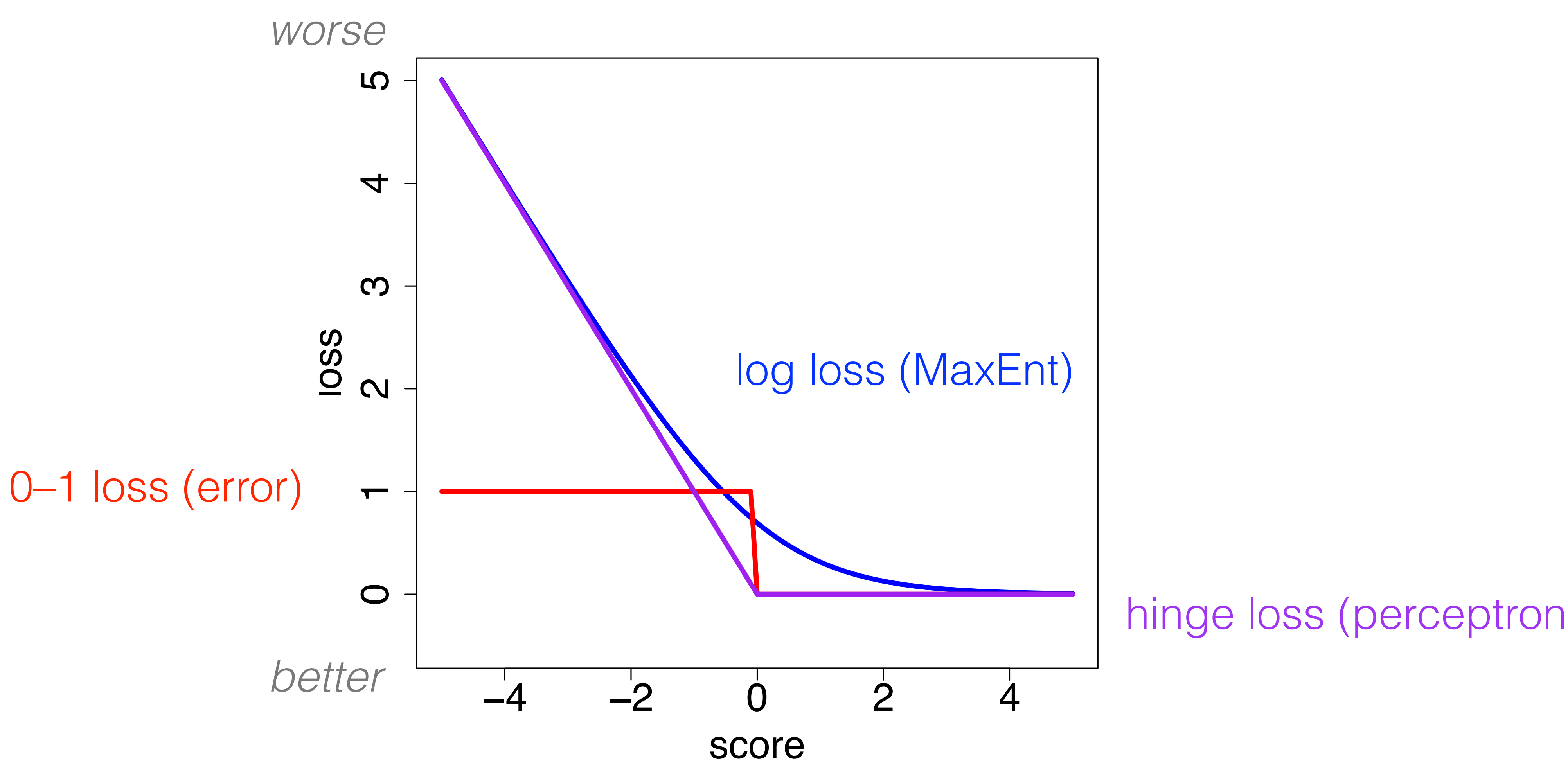
Objectives

- Naïve Bayes learning objective: **joint data likelihood**
  - $\mathbf{p}^* \leftarrow \arg\max_{\mathbf{p}} L_{joint}(\mathbf{D}; \mathbf{p})$
    $= \arg\max_{\mathbf{p}} \boldsymbol{\Sigma}_{(\mathbf{x},\, y) \in \mathbf{D}} \log \mathbf{p}(\mathbf{x}, y) = \arg\max_{\mathbf{p}} \boldsymbol{\Sigma}_{(\mathbf{x},\, y) \in \mathbf{D}} \log (\mathbf{p}(y)\mathbf{p}(\mathbf{x} \mid y))$
  - It can be shown that relative frequency estimation (i.e., count and divide, no smoothing) is indeed the maximum likelihood estimate
- MaxEnt learning objective: **conditional log likelihood**
  - $\mathbf{p}^* \leftarrow \arg\max_{\mathbf{p}} L_{cond}(\mathbf{D}; \mathbf{p})$
    $= \arg\max_{\mathbf{p}} \boldsymbol{\Sigma}_{(\mathbf{x},\, y) \in \mathbf{D}} \log \mathbf{p}(y \mid \mathbf{x})$

    $\mathbf{w} \leftarrow \arg\max_{\mathbf{w}} \boldsymbol{\Sigma}_{(\mathbf{x},\, y) \in \mathbf{D}} \mathbf{w}^{\top} \boldsymbol{\Phi}(\mathbf{x}, y) - \log \Sigma_{y'} \exp(\mathbf{w}^{\top} \boldsymbol{\Phi}(\mathbf{x}, y'))$ [2 slides ago]
  - This has no closed-form solution. Hence, we need an optimization algorithm to try different weight vectors and choose the best one.
  - With thousands or millions of parameters—not uncommon in NLP—it may also overfit.

Objectives

Visualizing different loss functions for binary classification



$\mathbf{w}^{\mathsf{T}} \Phi(\mathbf{x}, y')$

figure from Noah Smith

Objectives

- Why not just penalize error directly if that's how we're going to evaluate our classifier (accuracy)?
  - Error is difficult to optimize! Log loss and hinge loss are easier. Why?
    - Because they're differentiable.
    - Can use stochastic (sub)gradient descent (SGD) and other gradient-based optimization algorithms (L-BFGS, AdaGrad, ...). There are freely available software packages that implement these algorithms.
    - With supervised learning, these loss functions are **convex**: local optimum = global optimum (so in principle the initialization of weights doesn't matter).
    - The perceptron algorithm can be understood as a special case of subgradient descent on the hinge loss!
- N.B. I haven't explained the math for the hinge loss (perceptron) or the SVM. Or the derivation of gradients. See further reading links if you're interested.

A likelihood surface

Visualizes the likelihood objective (vertical axis) as a function of 2 parameters.
Likelihood = maximization problem. Flip upside down for the loss.

Gradient-based optimizers choose a point on the surface, look at its curvature, and then successively move to better points.

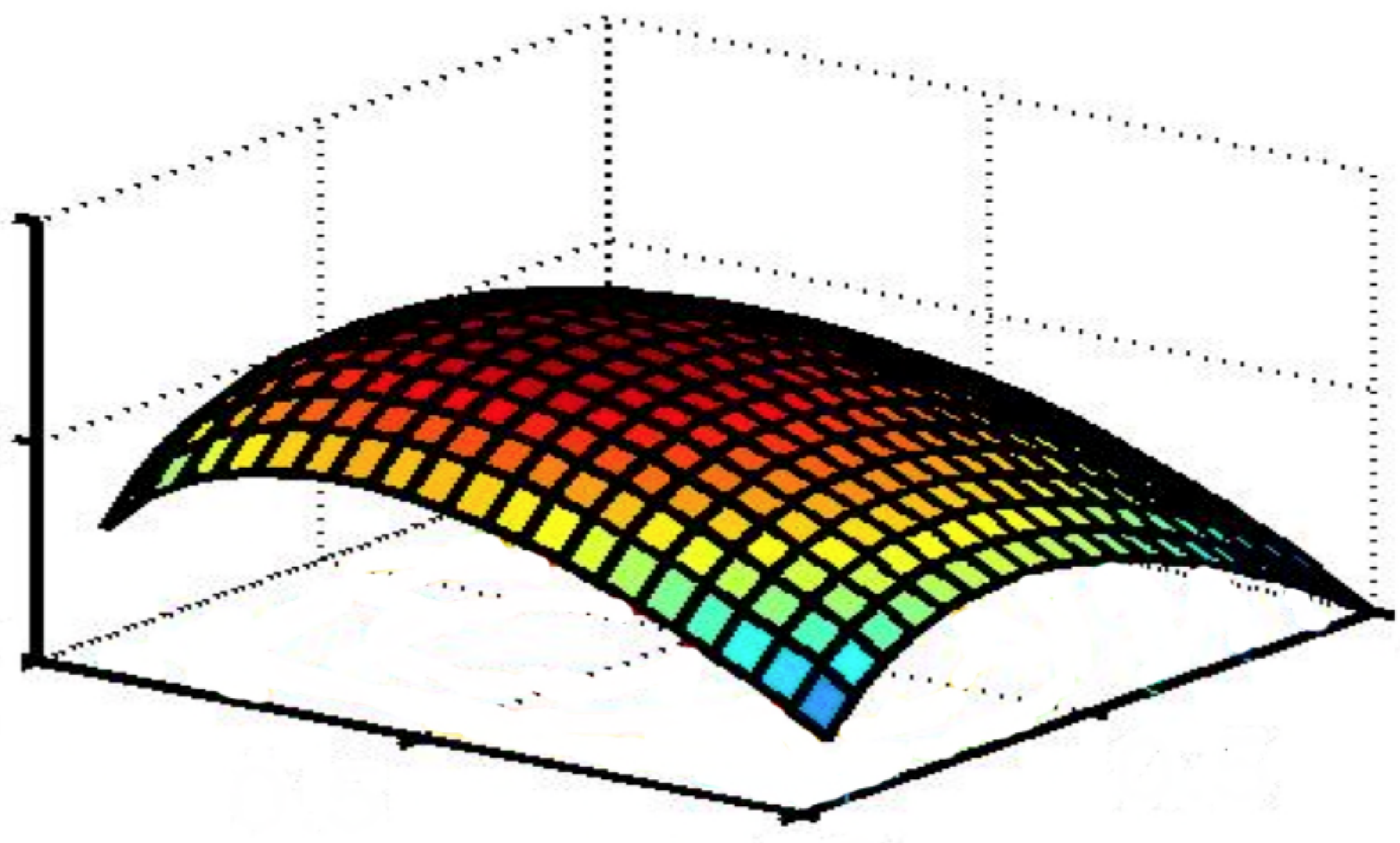


figure from Chris Manning

Regularization

- Better MaxEnt learning objective: **regularized conditional log likelihood**
  - $\mathbf{w}^* \leftarrow \arg\max_{\mathbf{w}} -\lambda R(\mathbf{w}) + \Sigma_{(\mathbf{x}, y) \in D}\, \mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y) - \log \Sigma_{y'} \exp(\mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y'))$
- **To avoid overfitting, the regularization term ("regularizer") $-\lambda R(\mathbf{w})$ penalizes complex models (i.e., parameter vectors with many large weights).**
  - Close relationship to Bayesian prior (a priori notion of what a "good" model looks like if there is not much training data). Note that the regularizer is a function of the weights only (not the data)!
- In NLP, most popular values of $R(\mathbf{w})$ are the ℓ_1 norm ("Lasso") and the ℓ_2 norm ("ridge"):
  - $\ell_2 = \|\mathbf{w}\|_2 = (\Sigma_i\, w_i^2)^{-1/2}$ encourages most weights to be **small in magnitude**
  - $\ell_1 = \|\mathbf{w}\|_1 = \Sigma_i\, |w_i|$ encourages most weights to be **0**
  - λ determines the tradeoff between regularization and data-fitting. Can be tuned on dev data.
- SVM objective also incorporates a regularization term. Perceptron does not (hence, averaging and early stopping).

Sparsity

- ℓ_1 regularization is a way to promote **model sparsity**: many weights are pushed to 0.
    - A vector is sparse if (# **nonzero** parameters) « (total # parameters).
    - Intuition: if we define very general feature templates—e.g. one feature per word in the vocabulary—we expect that *most features should not matter* for a particular classification task.
- In NLP, we typically have sparsity in our **feature vectors** as well.
    - E.g., in WSD, all words in the training data but *not* in context of a particular token being classified are effectively 0-valued features.
    - Exception: **dense** word representations popular in recent neural network models (we'll get to this later in the course).
- Sometimes the word "sparsity" or "sparseness" just means "not very much data."

Summary: Linear Models

Classifier: $y \leftarrow \arg\max_{y'} \mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}, y')$

	kind of model	*loss function*	*learning algorithm*	*avoiding overfitting*
Naïve Bayes	Probabilistic, generative	Likelihood	Closed-form estimation	Smoothing
Logistic regression (MaxEnt)	Probabilistic, discriminative	Conditional likelihood	Optimization	Regularization penalty
Perceptron	Non-probabilistic, discriminative	Hinge	Optimization	Averaging; Early stopping
SVM (linear kernel)	Non-probabilistic, discriminative	Max-margin	Optimization	Regularization penalty

Take-home points

- Feature-based linear classifiers are important to NLP.
  - You define the features, an algorithm chooses the weights. Some classifiers then exponentiate and normalize to give probabilities.
  - More features ⇒ more flexibility, also more risk of overfitting. Because we work with large vocabularies, not uncommon to have millions of features.
- Learning objective/loss functions formalize training as choosing parameters to optimize a function.
  - Some model **both** the language and the class (generative); some directly model the class *conditioned on* the language (discriminative).
  - In general: **Generative** ⇒ training is cheaper, but lower accuracy.
    Discriminative ⇒ higher accuracy with sufficient training data and computation (optimization).
- Some models, like naïve Bayes, have a closed-form solution for parameters. Learning is cheap!
- Other models require fancier optimization algorithms that may iterate multiple times over the data, adjusting parameters until convergence (or some other stopping criterion).
  - The advantage: fewer modeling assumptions. Weights can be interdependent.

Which classifier to use?

- Fast and simple: **naïve Bayes**
- Very accurate, still simple: **perceptron**
- Very accurate, probabilistic, more complicated to implement: **MaxEnt**
- Potentially best accuracy, more complicated to implement: **SVM**
- All of these: watch out for overfitting!
- Check the web for free and fast implementations, e.g. SVM^{light}

Further Reading: Basics & Examples

- **Manning:** features in linear classifiers
  http://www.stanford.edu/class/cs224n/handouts/MaxentTutorial-16x9-FeatureClassifiers.pdf

- **Goldwater:** naïve Bayes & MaxEnt examples
  http://www.inf.ed.ac.uk/teaching/courses/fnlp/lectures/07_slides.pdf

- **O'Connor:** MaxEnt—incl. step-by-step examples, comparison to naïve Bayes
  http://people.cs.umass.edu/~brenocon/inlp2015/04-logreg.pdf

- **Daumé:** "The Perceptron" (*A Course in Machine Learning*, ch. 3)
  http://www.ciml.info/dl/v0_8/ciml-v0_8-ch03.pdf

- **Neubig:** "The Perceptron Algorithm"
  http://www.phontron.com/slides/nlp-programming-en-05-perceptron.pdf

Further Reading: Advanced

- **Neubig:** "Advanced Discriminative Learning"—MaxEnt w/ derivatives, SGD, SVMs, regularization
  http://www.phontron.com/slides/nlp-programming-en-06-discriminative.pdf

- **Manning:** generative vs. discriminative, MaxEnt likelihood function and derivatives
  http://www.stanford.edu/class/cs224n/handouts/MaxentTutorial-16x9-MEMMs-Smoothing.pdf, slides 3–20

- **Daumé:** linear models
  http://www.ciml.info/dl/v0_8/ciml-v0_8-ch06.pdf

- **Smith:** A variety of loss functions for text classification
  http://courses.cs.washington.edu/courses/cse517/16wi/slides/tc-intro-slides.pdf & http://courses.cs.washington.edu/courses/cse517/16wi/slides/tc-advanced-slides.pdf

Evaluating Multiclass Classifiers and Retrieval Algorithms

Accuracy

- Assume we are disambiguating word senses such that every token has 1 gold sense label.
- The classifier predicts 1 label for each token in the test set.
- Thus, every test set token has a predicted label (*pred*) and a gold label (*gold*).
- The **accuracy** of our classifier is just the % of tokens for which the predicted label matched the gold label: $\#_{pred=gold}/\#_{tokens}$

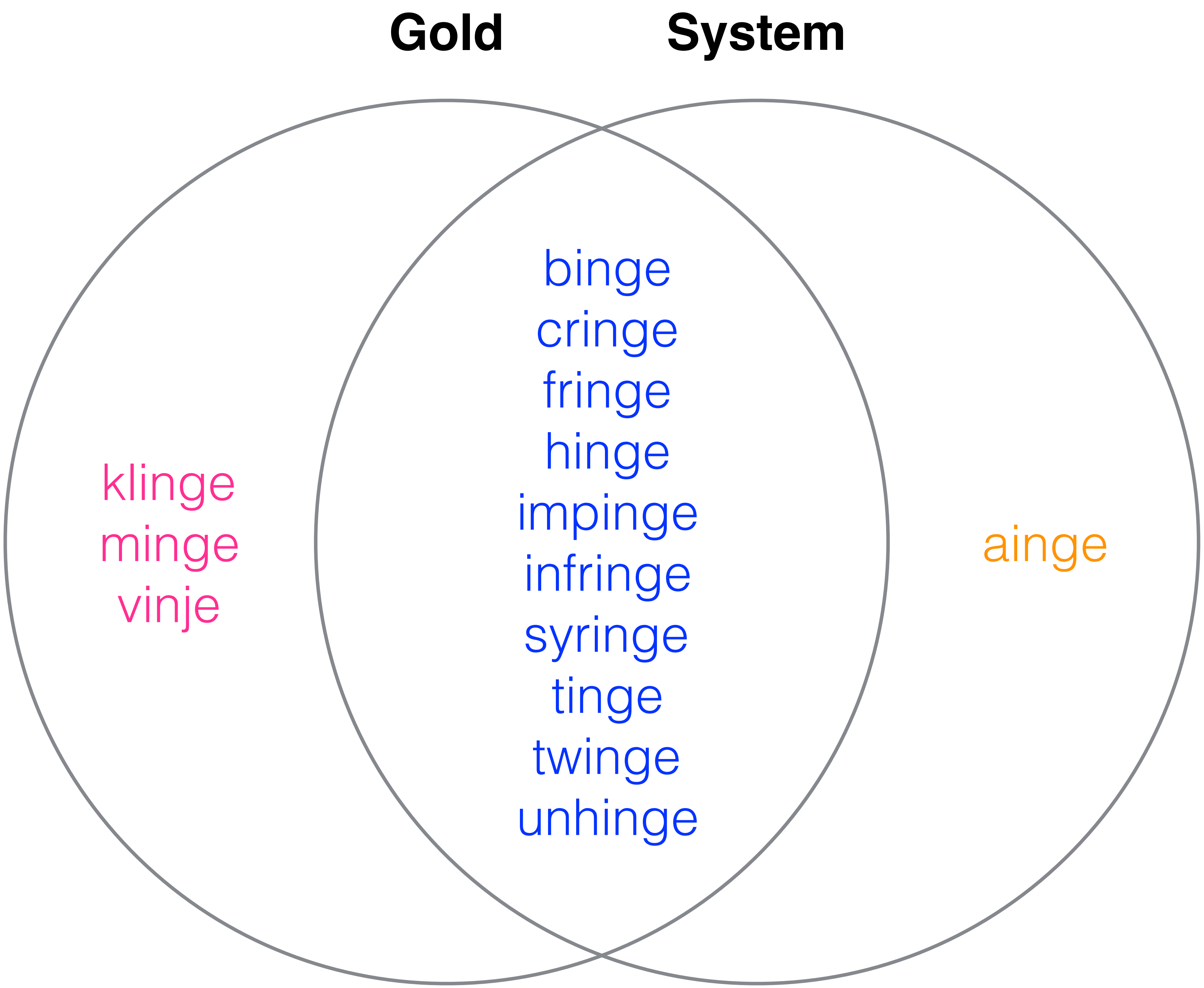
Precision and Recall

- To measure the classifier with respect to a certain label *y*, and there are >2, we distinguish precision and recall:
  - **precision** = proportion of times the label was predicted and that prediction matched the gold: $\#_{pred=gold=y} / \#_{pred=y}$
  - **recall** = proportion of times the label was in the gold standard and was recovered correctly by the classifier: $\#_{pred=gold=y} / \#_{gold=y}$
- The harmonic mean of precision and recall, called **F_1-score**, balances between the two.
  F_1 = 2*precision*recall / (precision + recall)

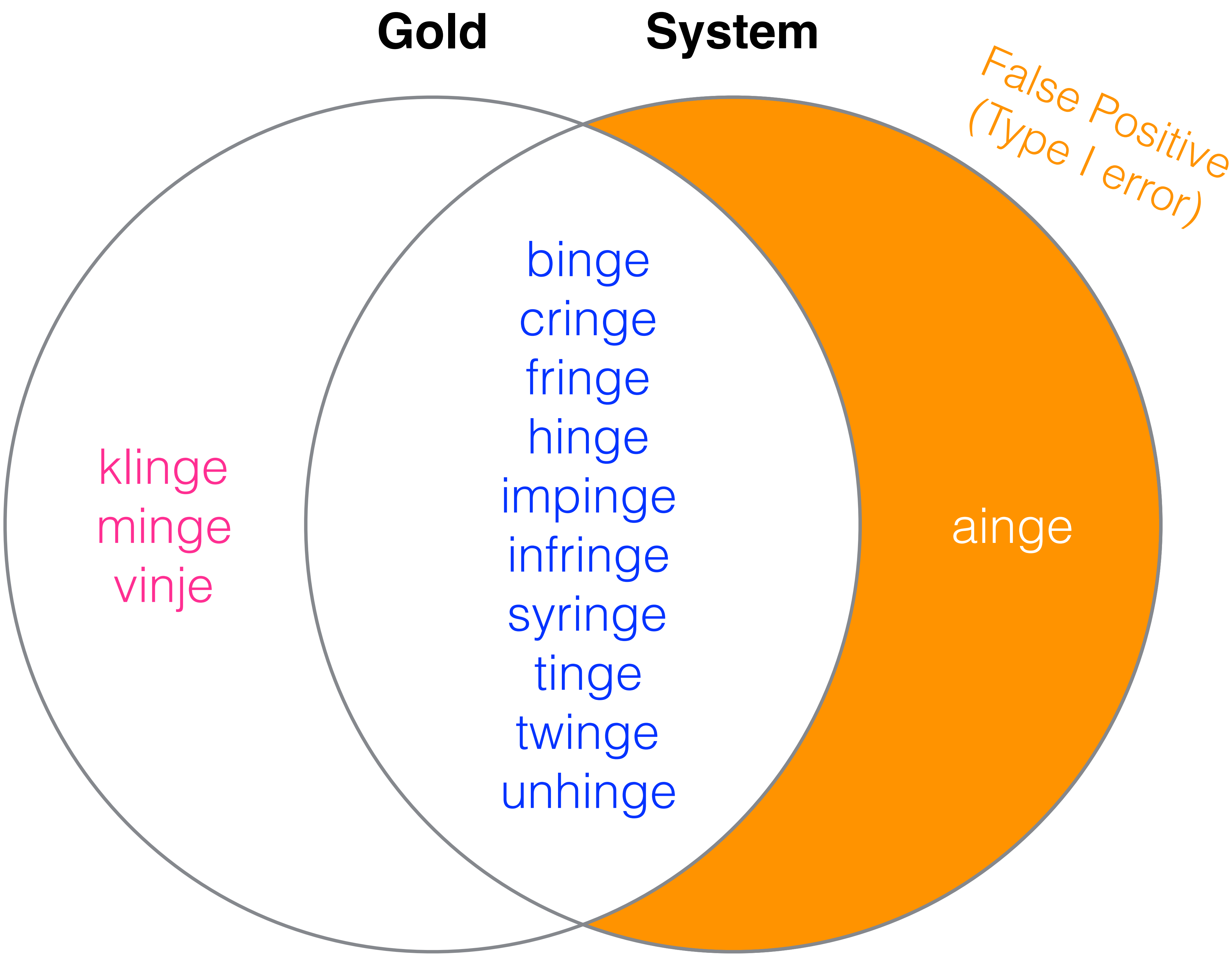
Evaluating Retrieval Systems

- Precision/Recall/F-score are also useful for evaluating retrieval systems.
- E.g., consider a system which takes a word as input and is supposed to retrieve all rhymes.
- Now, for a single input (the query), there are often many correct outputs.
- **Precision** tells us whether most of the given outputs were valid rhymes; **recall** tells us whether most of the valid rhymes in the gold standard were recovered.

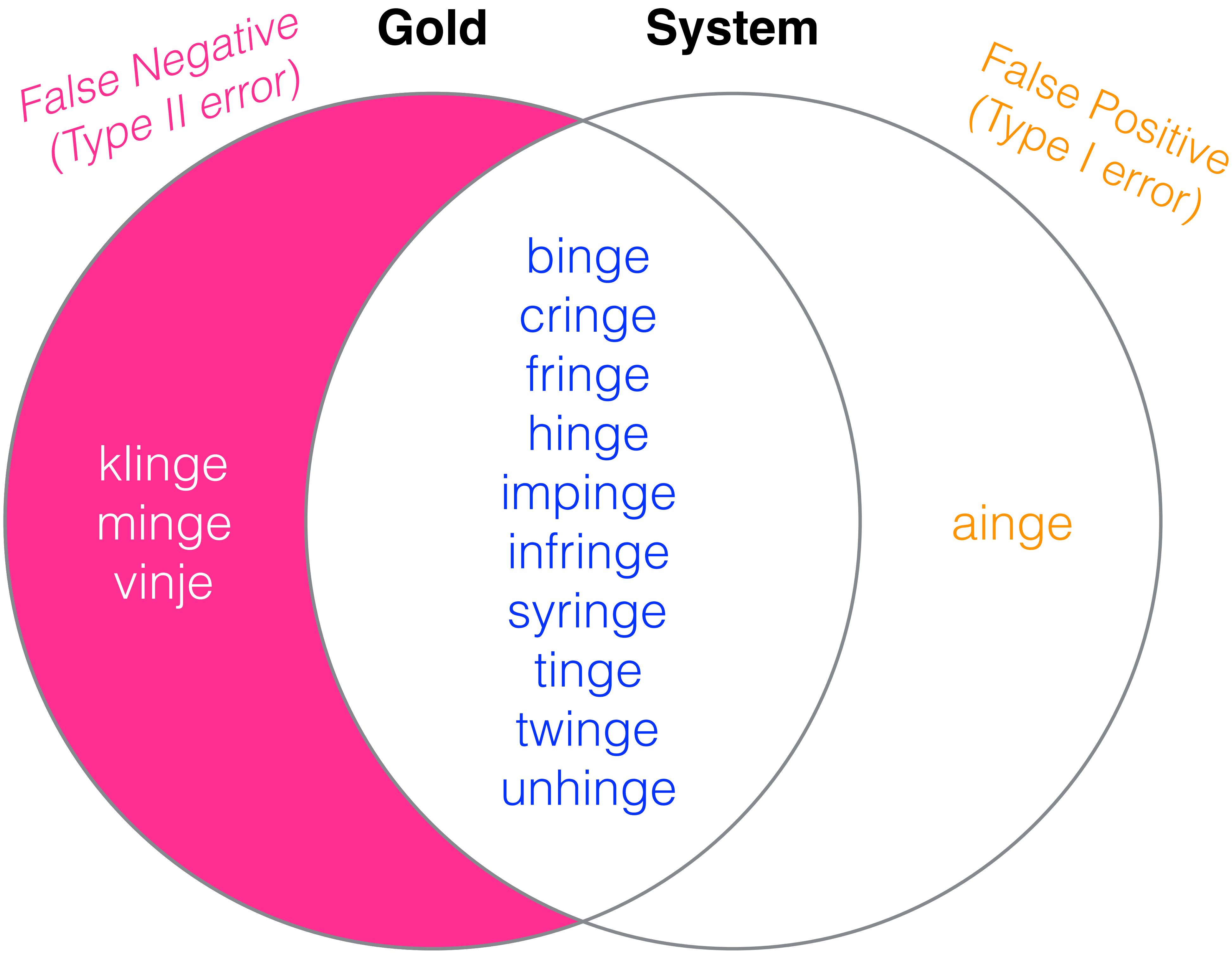
Rhymes for “hinge”



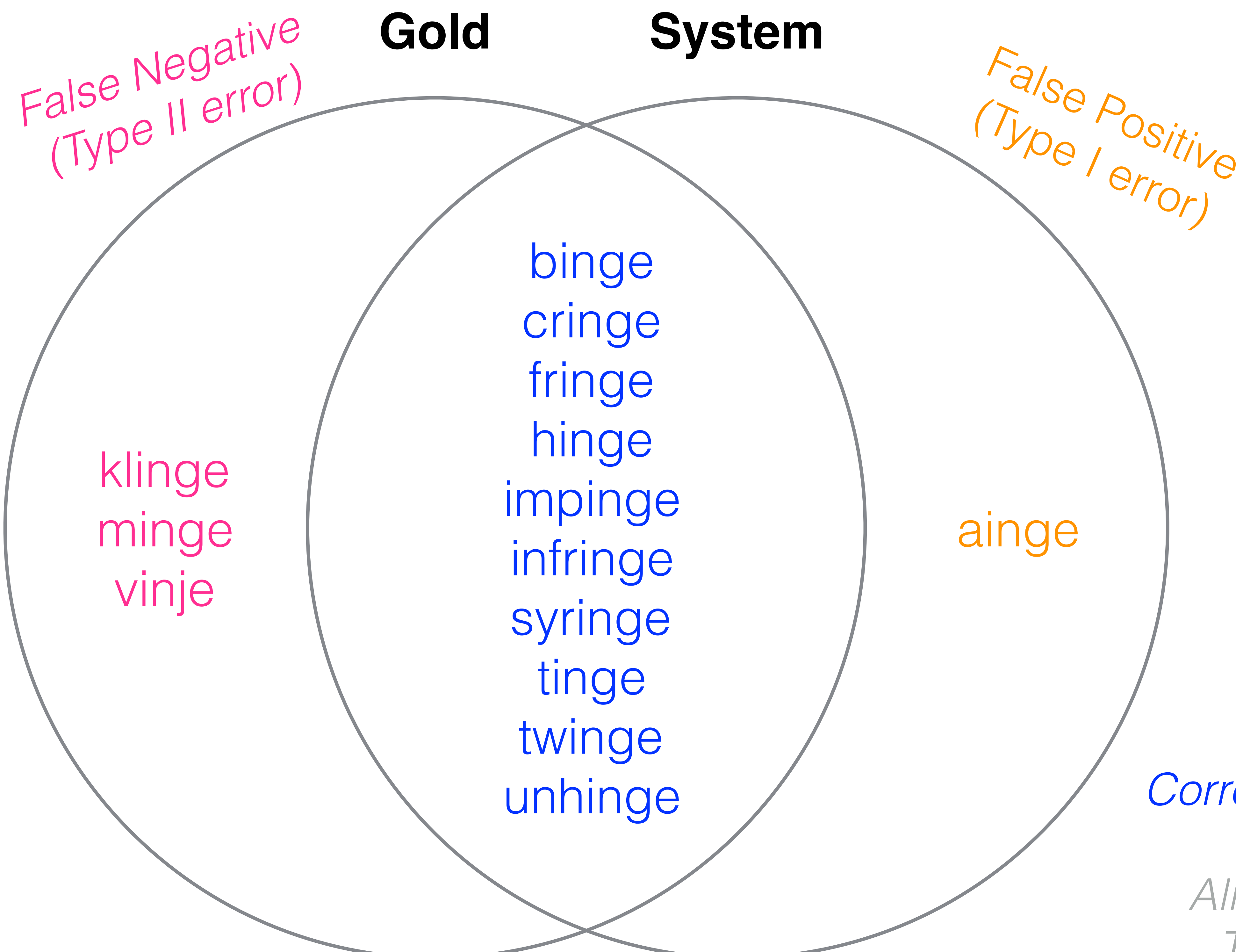
Rhymes for “hinge”



Rhymes for "hinge"



Rhymes for "hinge"

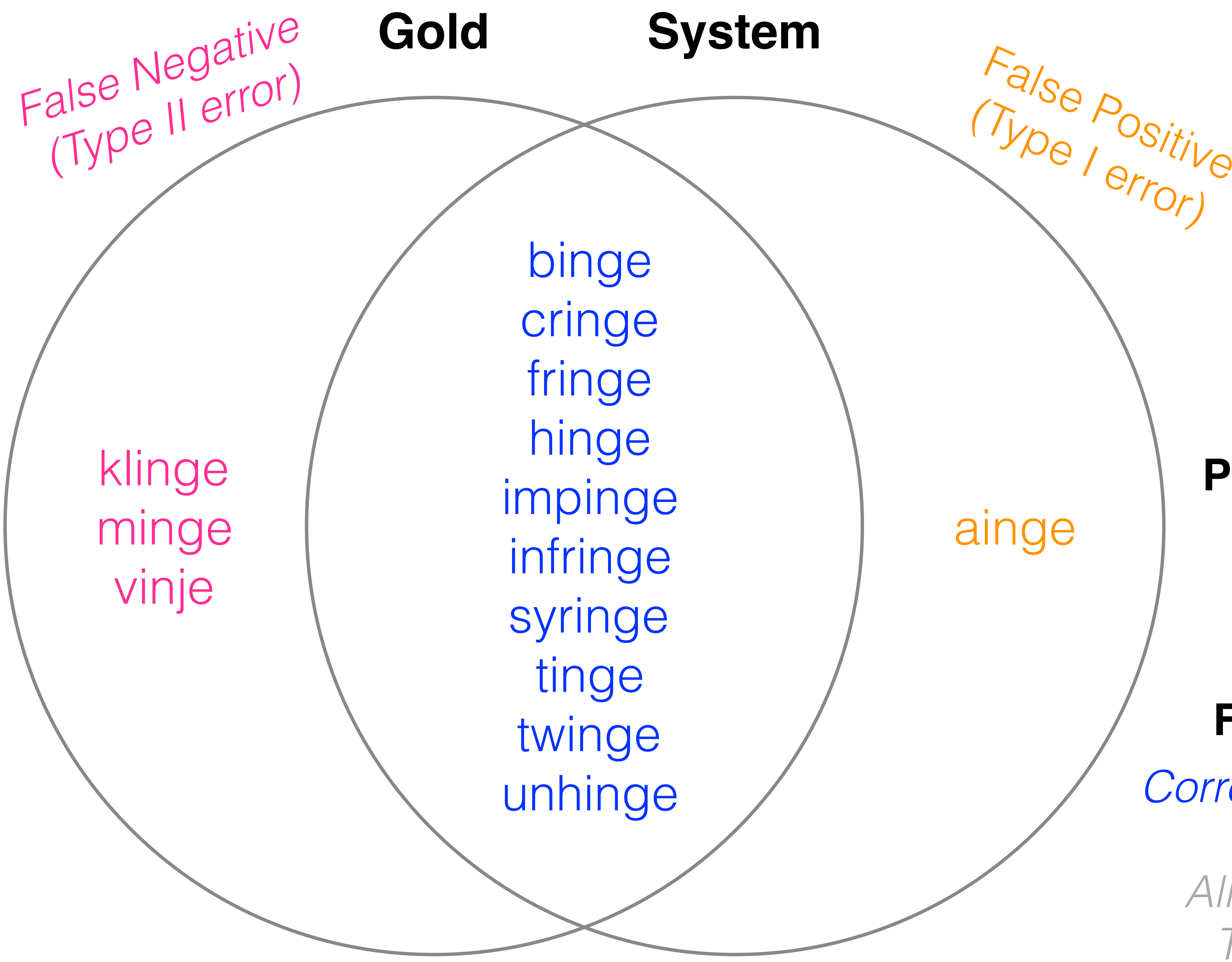


	Sys=Y	Sys=N
Gold=Y	10	3
Gold=N	1	(large)

Correctly predicted = True Positive

All other words = True Negative

Precision & Recall



	Sys=Y	Sys=N
Gold=Y	10	3
Gold=N	1	(large)

Precision = TP/(TP+FP)
= 10/11 = 91%

Recall = TP/(TP+FN)
= 10/13 = 77%

F_1 = 2·P·R/(P+R) = 83%

Correctly predicted = True Positive

All other words = True Negative