

Linear Models for Classification: Features & Weights

Nathan Schneider
(some slides borrowed from Chris Dyer)
ENLP | 4 January 2025

Outline

- **Words, probabilities → Features, weights**

this lecture

- **Geometric view: decision boundary**

next lecture

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

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*: $p(y \mid \mathbf{x}) \propto p(y) \cdots p(\text{Wall} \mid y) p(\text{Street} \mid y) p(\text{Wall Street} \mid y)$

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

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

	$\phi(\mathbf{x})$
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

...

- Turns the input into a table of features with real values (often binary: 0 or 1).

Feature Extraction

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

$\phi(x)$

spelling feature

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

...

- Turns the input into a table of features with real values (often binary: 0 or 1).

Feature Extraction

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

$\phi(x)$

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).

...

Feature Extraction

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

$\phi(x)$

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

	$\phi(x)$
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

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

$\phi(x)$

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

	$\phi(x)$
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

bias feature (\approx 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

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

\mathbf{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 $\phi(\mathbf{x})$ 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:

$$\begin{aligned} & \mathbf{w}_{y'}^\top \phi(\mathbf{x}) \\ &= \sum_j w_{y',j} \cdot \phi_j(\mathbf{x}) \end{aligned}$$

- For binary classification, equivalent to: $\text{sign}(\mathbf{w}^\top \phi(\mathbf{x}))$ — **+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**.
 - $\mathbf{w}^T \phi(\mathbf{x}) = 6.59$, $\mathbf{w}^T \phi(\mathbf{x}') = -6.74$

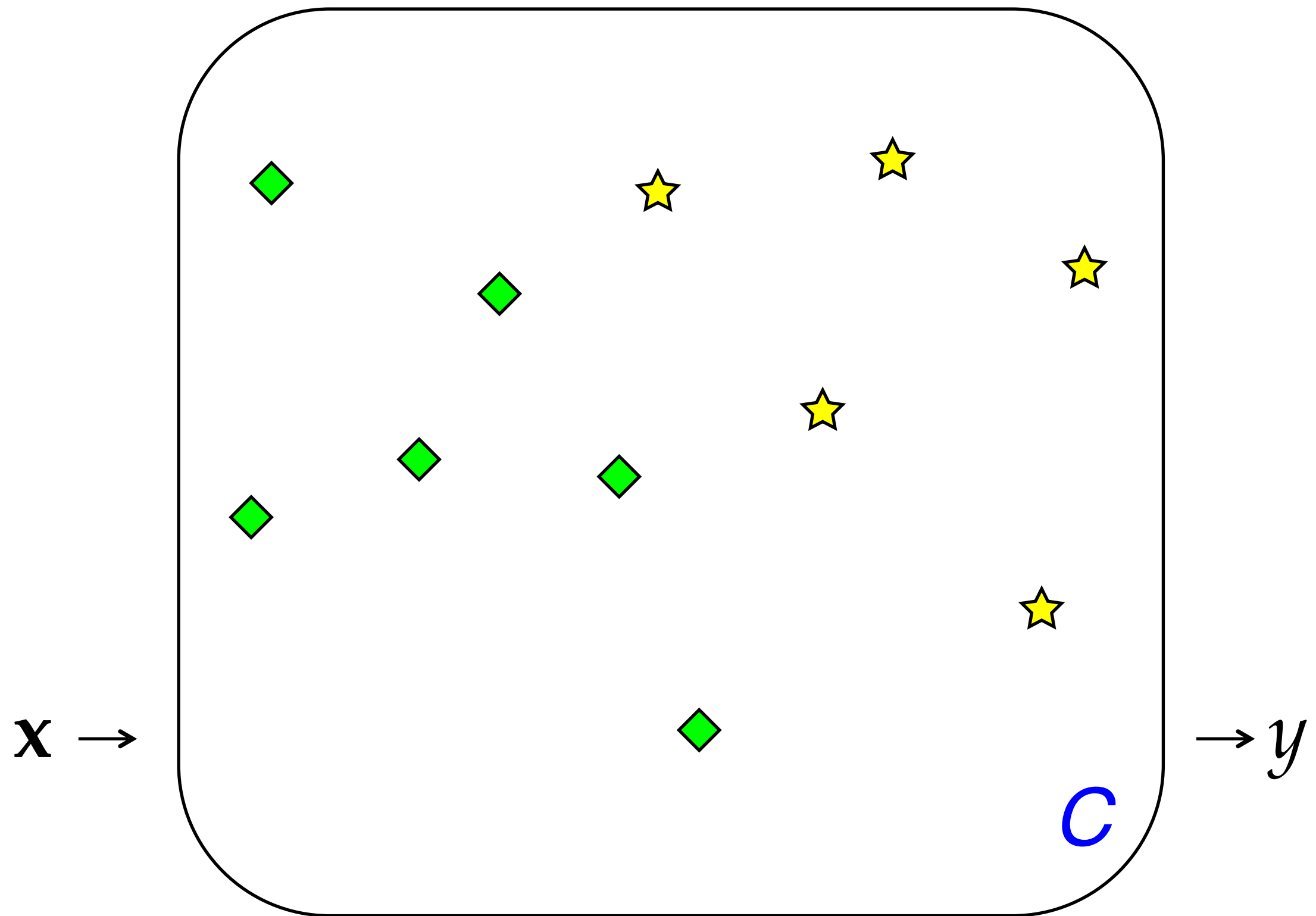
More than 2 classes

- Simply keep a separate weight vector for each class: \mathbf{w}_y
- The class whose weight vector gives the highest score wins!

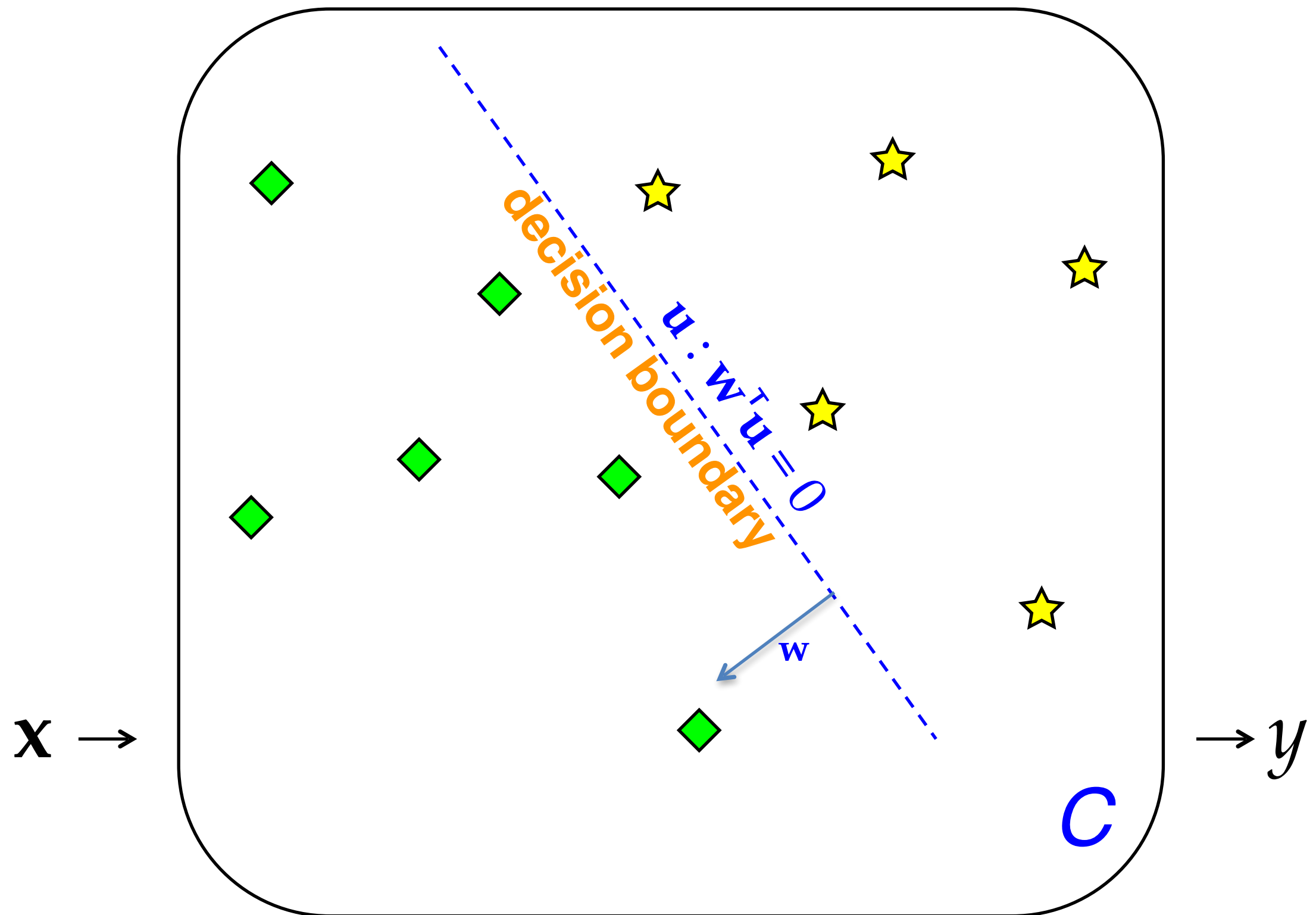
Learning the weights

- Weights depend on the choice of model and learning algorithm.
- Naïve Bayes fits into this framework, under the following estimation procedure for \mathbf{w} :
 - $w_{\text{bias}} = \log p(y)$
 - \forall features f : $w_f = \log p(f \mid y)$
 - $$\begin{aligned}\sum_j w_j \cdot \phi_j(\mathbf{x}) &= w_{\text{bias}} + \sum_{f \in \phi(\mathbf{x})} w_f \\ &= \log p(y) + \sum_{f \in \phi(\mathbf{x})} \log p(f \mid y) \\ &= \log (p(y) \cdot \prod_{f \in \phi(\mathbf{x})} p(f \mid y))\end{aligned}$$
- 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**.

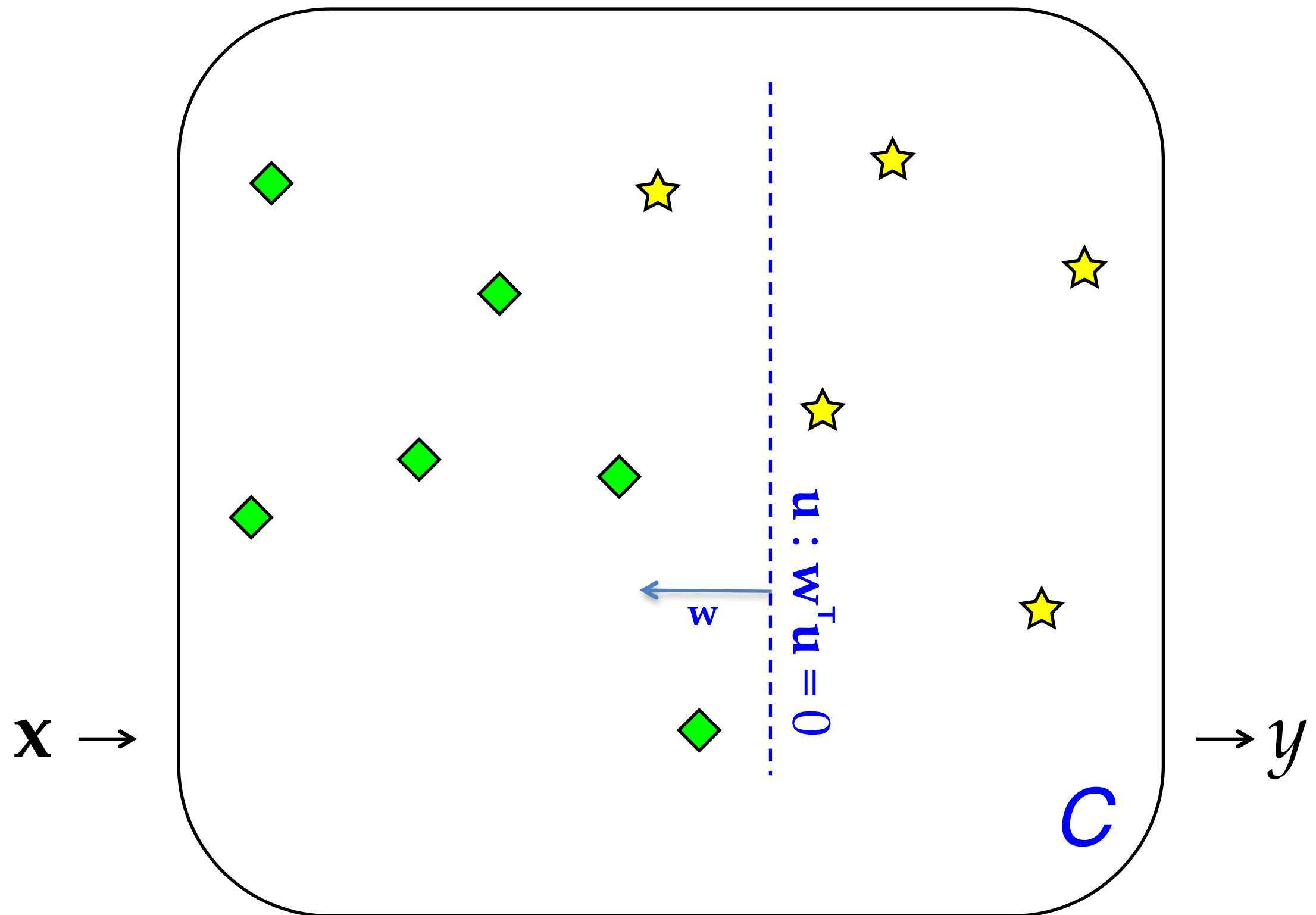
Linear Classifiers: Geometric View



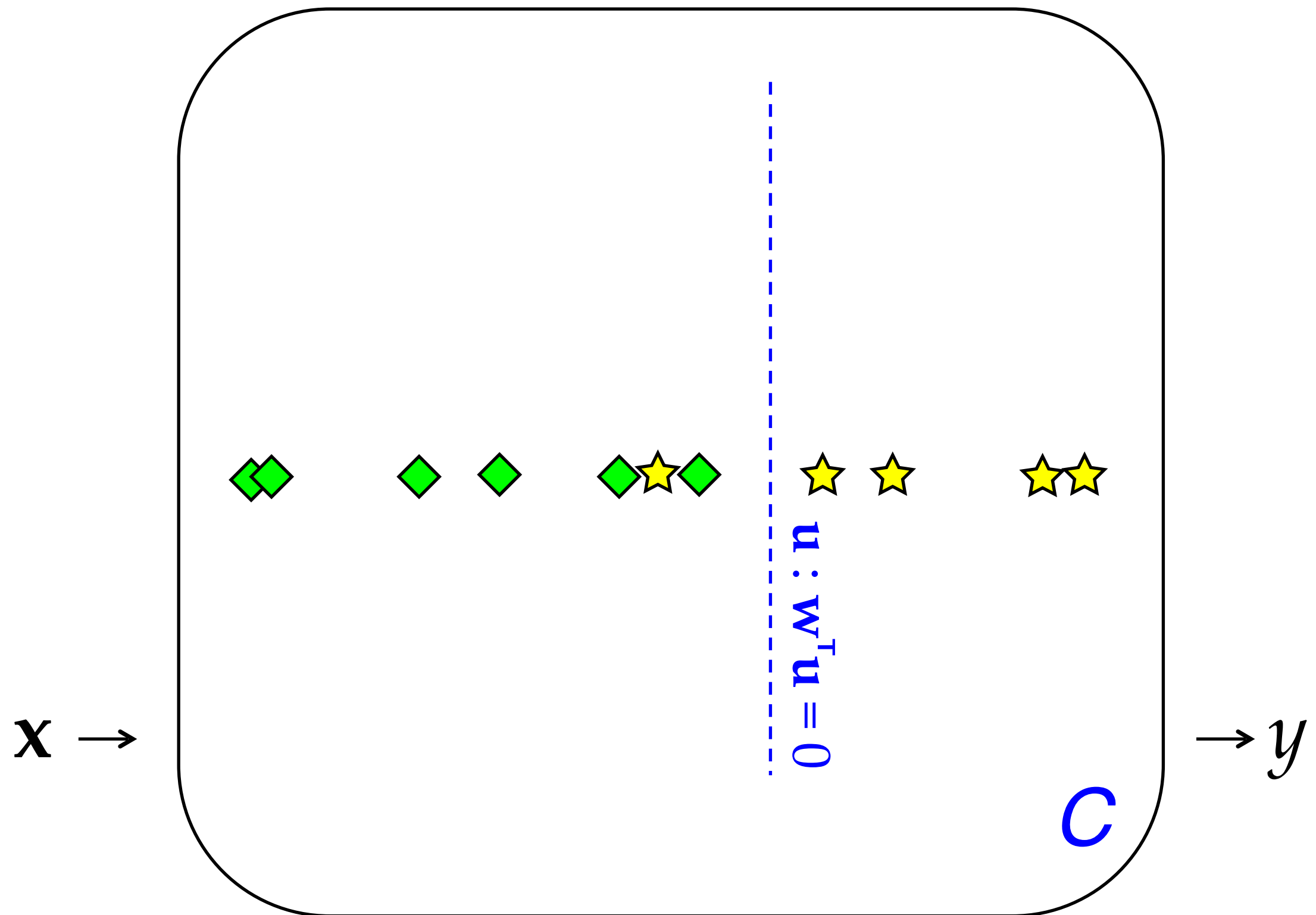
Linear Classifiers: Geometric View



Linear Classifiers: Geometric View



Linear Classifiers: Geometric View



Linear Classifiers (> 2 Classes)

return

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

$\mathbf{x} \rightarrow$

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 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 $\wedge y=VERB$**

