Supervised Learning for NLP I

HSS 510 / DS 518: NLP for HSS

Taegyoon Kim

Mar 18, 2025

Agenda

Things to be covered

- Overview of supervised learning
- Step 1: Building a labeled data set
- Step 2: Training models
- Step 3: Evaluating performance
- Guided labeling: text classification with movie reviews data in Python

Supervised Learning

Supervised vs. Unsupervised

	Supervised	**Unsupervised**
Objective	Trained on a labeled data to learn a mapping from input to output	Find patterns or structures within data without labels
Outcome	Pre-defined categories	Not quite pre-defined
Model evaluation	Explicit metrics such as accuracy, precision, recall, or MSE	Can involve qualitative assessment
Examples	Classification/regression for texts	Topic models

Supervised Learning

Regression vs. Classification

- Regression
  - The outcome of interest is continuous or ordered (beyond binary)
  - E.g., OLS regression (+ non-linear regression algorithms such as random forest regression)
- Classification
  - The outcome is a value in an unordered set (i.e. categories)
- The two approaches share the broad principles of supervised leaning and can be adapted

Supervised Learning

We will focus on text classification with supervised learning

- Goal
  - To classify documents into pre-defined categories
  - E.g., sentiment of comments (e.g., positive-negative), stance on issues (e.g., for-against-neutral), etc.
- We need
  - Labeled data set (for training and testing)
  - Model (algorithm) that maps texts to labels
  - Evaluation approaches: performance metrics, cross-validation, etc.

Supervised Learning

Evolution of text classification

- Dictionary methods
    - Based on counting/weighting of relevant keywords
    - Readily available and fast ↔ sub-optimal performance
    - If you want to quick, preliminary analysis for concepts supported by an existing dictionary, it can be helpful
    - E.g., LIWC, VADER, Moral Foundations Dictionaries, etc.

Supervised Learning

Evolution of text classification (cont'd)

- Traditional ML algorithms
  - This approach laid the groundwork for many foundational concepts in text classification
  - Classifiers (models) are trained to learn the relationships between texts and labels (i.e., classes)
  - So this requires training (labeled) data (pairs of a text and a label)
  - (On average) more training data, higher performance
  - E.g., Logistic regression, random forest, SVM, deep neural networks

Supervised Learning

Evolution of text classification (cont'd)

- Fine-tuning representation models (e.g., BERT family)
  - These models are pre-trained with massive amounts of text data
  - Given a text, representation models encode it into a vector (an array of numbers) that captures its meaning
  - We can fine-tune such a model for classification with potentially higher performance
  - They tend to achieve higher performance than the traditional ML approaches
  - Week 13 will cover this approach

Supervised Learning

Evolution of text classification (cont’d)

- Prompting generative models (e.g., GPT family)
  - Like representation models, these models are pre-trained on vast amounts of text data, often even more extensively
  - Generative models are designed to generate text outputs given input text prompts
  - We can prompt such a model with unlabeled texts to generate labels
  - Still requires labeled data (not for training but for evaluating performance)
  - It is also possible to “fine-tune” these models
  - Weeks 14–15 will cover this

Overview of Process

Overall process

- Step 1: building a labeled data set
- Step 2: training model(s)
- Step 3: evaluating performance

Overview of Process

Overall process

- **Step 1: building a labeled data set**
- Step 2: training model(s)
- **Step 3: evaluating performance**

∗ Steps 1 and 3 apply to all of the classification approaches

Overview of Process

Step 1: build a labeled data set

- Label texts following systematic labeling guidelines and check inter-coder reliability
- This (C) will serve as “ground-truth” or “gold standards”
- C is used to training (building a model) and validation (evaluating performance)

Overview of Process

Step 1: build a labeled data set

Doc number	Text	y
1	This is great!	0
2	%@% off!	1
...		
9999	This is sick	0
10000	Love BTS <3	0

Overview of Process

Step 1: build a labeled data set

Doc number	Text	y
1	This is great!	0
2	%@% off!	1
...		
9999	This is sick	0
10000	Love BTS <3	0

Overview of Process

Step 2: train models

- Randomly split C into a training set (C_{train}) and a test set (C_{test})
  - Typically, $C_{train} : C_{test}$ = 7:3 or 8:2
  - E.g., identifying YouTube comments containing hate speech
    - C: 10,000 comments labeled for the presence of hate speech
    - $C_{training}$: 8,000 comments for training
    - C_{test}: 2,000 comments for test
- Generate X_{train} (feature matrix) from C_{train}
  - E.g., count vectors, TF-IDF, or embeddings

Overview of Process

Step 2: train models

Index	y	Token 1	Token 2	…	Token V-1	Token V
1	0	3	1.4	…	1.7	6
2	1	-0.8	6.4	…	5.7	-1.6
…						
7999	0	-2.8	0.9	…	3.3	-0.6
8000	0	3.7	1.4	…	5.7	-5.8

Overview of Process

Step 2: train models

- Choose a model F (e.g., logistic regression) and learn model parameters β (e.g., an array of coefficients)
  - The model provides a mapping between X_{train} and y_{train}
- Loss (cost) function: measures how much model predictions ($\hat{y}_{train}$) differ from the true labels (y_{train})
  - $\hat{y}_{train} = F(\hat{\beta} * X_{train})$
  - β is estimated in a way that minimizes the difference between the two
- As a result, we get a classifier: $\hat{y} = F(\hat{\beta} * X)$

Overview of Process

Step 2: train models

Index	y	$\hat{y}$	Token 1	Token 2	…	Token V-1	Token V
1	0	1	3	1.4	…	1.7	6
2	1	1	-0.8	6.4	…	5.7	-1.6
…							
7999	0	0	-2.8	0.9	…	3.3	-0.6
8000	0	0	3.7	1.4	…	5.7	-5.8

Overview of Process

Step 3: evaluate performance

- We held out another labeled set C_{test} (n = 2,000) (**why?**)
- Use the classifier $F(\hat{\beta} * X)$ to generate predictions $\hat{y}_{test}$
- Compare the predictions $\hat{y}_{test}$ and the true labels y_{test}
- Performance metrics include accuracy, precision, recall, etc.
- (Then use the classifier for unlabeled data)

Overview of Process

Step 3: evaluate performance

Index	y	$\hat{y}$	Token 1	Token 2	…	Token V-1	Token V
1	1	1	3.12	1.99	…	5.77	0.36
2	1	0	-0.8	1.14	…	9.71	-1.66
…							
1999	0	0	-2.11	0.95	…	1.23	-0.62
2000	0	0	3.71	1.48	…	1.7	-5.84

Bias, Variance, and Overfitting

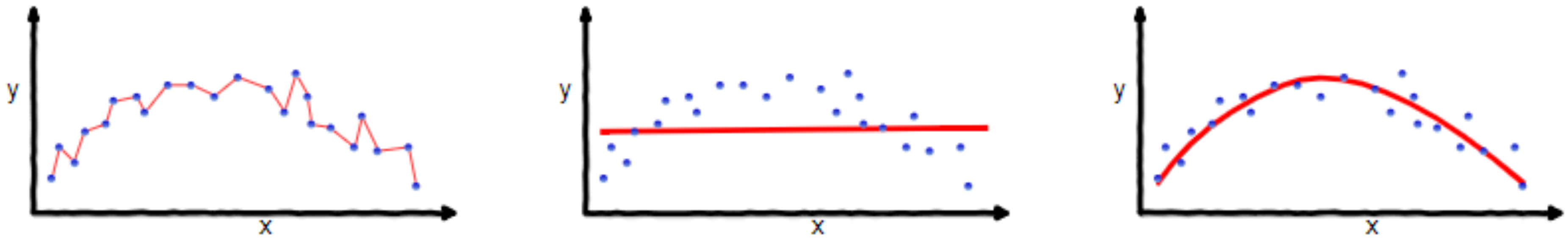
Bias

- The degree to which the model's predictions deviate from the true labels in a systematic manner
- A model with high bias make predictions that are consistently off-target

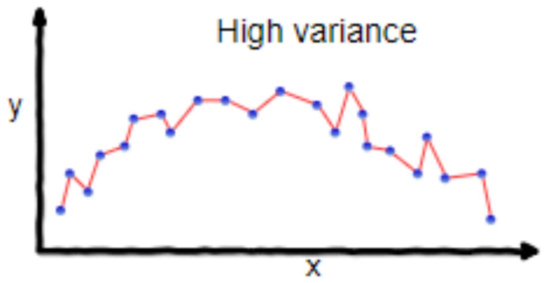
Variance

- The degree to which the model generalizes to different data
- High variance means low generalizability

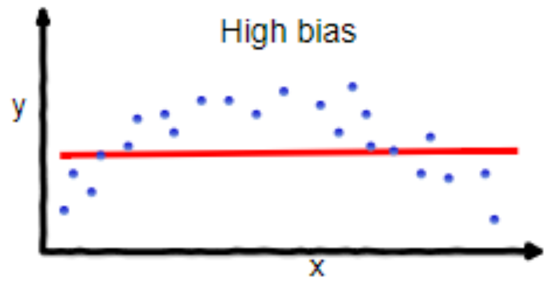
Bias, Variance, and Overfitting



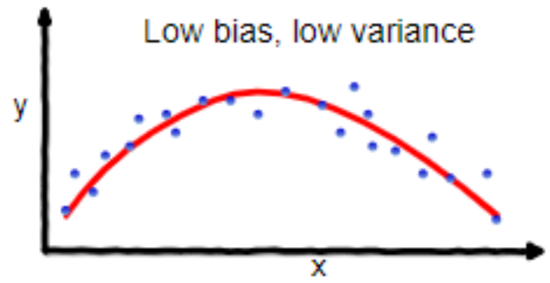
Bias, Variance, and Overfitting



overfitting



underfitting

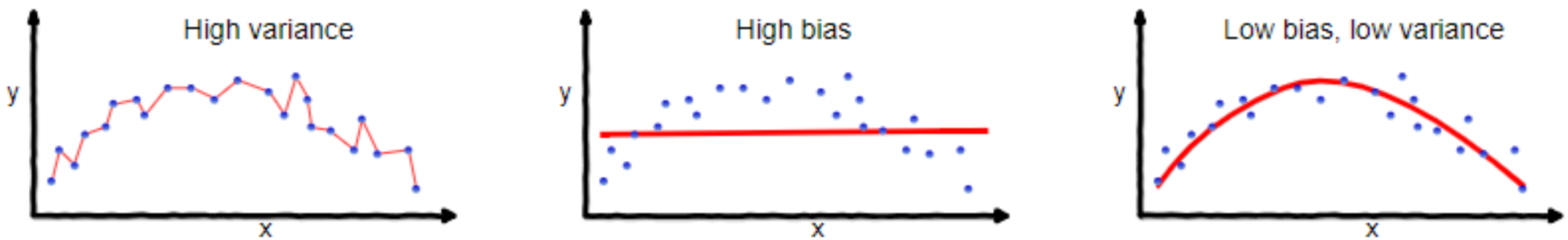


Good balance

Bias, Variance, and Overfitting

Overfitting and underfitting

- If a model learns the training data “too well” (low bias), it can lead to overfit
- This happens when the model mistakes noise for signal
- The model would not generalize to the test set (high variance)



overfitting underfitting Good balance

Bias, Variance, and Overfitting

Training-test split

- A minimal measure to prevent overfitting
- The primary goal here is to make our model as generalizable as possible
- “Generalizable” means being able to perform well on unseen documents (other than the documents the model was trained on)
- When a model learns the noise or random fluctuations in the training set, this typically results in a model that performs poorly on new the test set

Step 1: Building a Labeled Data Set

In HSS and many other fields, text classification is often used to “measure” concepts using texts

- Some concepts are already available in the form of labeled texts: e.g., news categories
- In many cases, however, concepts we are trying to measure using texts need to be measured from scratch
- The process involves definition/conceptualization and manual labeling

Step 1: Building a Labeled Data Set

How do we build a labeled data set?

- A form of manual content analysis
- Here human labels are often considered “ground truth” or “gold standard”
- Manually labeled data are used for training (train set) and evaluation (test set)

Step 1: Building a Labeled Data Set

How do we build a labeled data set?

- Expert labeling
  - In many projects, a few domain experts work on labeling
    - E.g., a researcher + two RAs
  - Define the concept and draft coding guidelines
  - Annotators should be trained to learn the concept and related guidelines

Step 1: Building a Labeled Data Set

How do we build a labeled data set? (cont'd)

- Crowd-sourced labeling
  - "Wisdom of crowd": aggregated judgments of (online) non-experts converge to judgments of experts at much lower cost (Benoit et al, 2016)
  - Difficult to educate annotators on sophisticated tasks
  - Inductive measurement based on a relatively loose conceptualization

Step 1: Building a Labeled Data Set

Expert labeling vs. Crowd Sourcing

- Deductive vs. inductive
- Degree of training
- Scalability (cost)

Step 1: Building a Labeled Data Set

Selecting texts for manual labeling

- The selected texts should be representative of the entire corpus.
- These texts are used to train and test your model
- Mismatches between the chosen texts and the texts the model is applied to can reduce generalizability
- E.g., changes in anti-vaccine discourse throughout 2020
    - After mid-2020, anti-vaccine discourse increasingly focused on government mandates
    - If the model is trained only on texts from the first two months of 2020, it may fail to capture these later shifts

Step 1: Building a Labeled Data Set

Iterative process

- The process of building a labeled set does not often take place at once but in an iterative process
- In many cases, specifying comprehensive annotation guidelines *ex ante* can be challenging
- Preliminary labeling rules are written and applied to an initial set of texts → annotators identify disagreements in their labels and ambiguities in the rules → revision of the rule → manual annotation of another set of texts …

Step 1: Building a Labeled Data Set

Dealing with subjectivity

- Many concepts in humanities and social sciences tend to subjective
- This is, from the beginning, why 1) careful conceptualization and 2) writing clear labeling rules, and 3) training coders are extremely important
- Once your final labels are built, it is helpful to assess reliability between annotators (inter-coder reliability)
    - Krippendorf's α, Cohen's κ (alternatives include Pearson's **r**, Spearman's ρ)
    - Relevant tools: **krippendorff** in Python or **irr** in R

Step 1: Building a Labeled Data Set

Who are the annotators?

- Expert coding
  - Academics/students (Javdani and Chang 2023)
- Crowdsourcing
  - Skewed distribution of worked hours (Difallah et al. 2018)
  - Inattentive workers (Peyton et al. 2022; Ternovski 2022)
  - LLM-based responses (Veselovsky et al. 2023)
  - Demographic characteristics (Al Kuwatly et al. 2020)

Step 2: Training Models

We the need to generate features

- Features mean numerical representations of text data that models can process
- $C_{train} \rightarrow \quad X_{train}$
- Options include count vectors, TF-IDF vectors, word/document embeddings, etc.
- Note, as we will see in later weeks, we hardly need manual feature extraction when we fine-tune or prompt pre-trained models

Step 2: Training Models

We the need to generate features (cont'd)

Index	Token 1	Token 2	…	Token V-1	Token V
1	3	1.4	…	1.7	6
2	-0.8	6.4	…	5.7	-1.6
…					
7999	-2.8	0.9	…	3.3	-0.6
8000	3.7	1.4	…	5.7	-5.8

Step 2: Training Models

So far we have:

- Built a labeled data set
- Generated a feature matrix
- This means that we have the outcome (y) and features (X_{train})

Now we will:

- Select a model F
- Learn model parameters β to build a classifier ($\hat{y} = F(\hat{\beta} * X)$)

Step 2: Training Models

Numerous algorithms

- Logistic regression
- Naive Bayes
- Support vector machine
- Tree-based models (decision tree, random forest, XGBoost, etc.)
- Neural networks
- Etc.

Step 2: Training Models

Logistic regression

- Used to classify a document into binary categories
  - For more than two categories, multinomial logistic regression
- One of the most useful analytics tools in science (not just NLP/ML)
- The baseline supervised learning algorithm for classification
- Forms the basis of neural networks

Step 2: Training Models

Components of classification with logistic regression

- Features
  - A document is represented as a vector of features $\vec{x}$ (= $[x_1, \ldots, x_n]$)
- A classification function (F)
  - $p(y = 1|x)$ is computed for each document given the feature vector ($\vec{x}$) and the parameters (β: $\vec{w}$ and b)
  - The sigmoid function transforms $p(y = 1|x)$ into a value between 0 & 1
- Loss function (measures how model prediction ŷ is different from y during training) and an algorithm for optimizing it (gradient descent)

Step 2: Training Models

How does logistic regression compute predicted probabilities?

- $p(y = 1|x)$
  - The probability of y = 1 given a feature vector $\vec{x}$ (= $[x_1, \ldots, x_n]$)
  - E.g., for a simple count vector, it would be # of times each token appears in the document
- Logistic regression learns β, a vector of coefficients
  - A bias term b: a single number (a.k.a. intercept)
  - Weights $\vec{w}$ (= $[w_1, \ldots, w_n]$)
    - E.g., tokens signaling hateful intention would get high weights
  - With b, $\vec{w}$, and $\vec{x}$, we compute z = $(\sum_{i=1}^{n} w_i x_i)$ + b

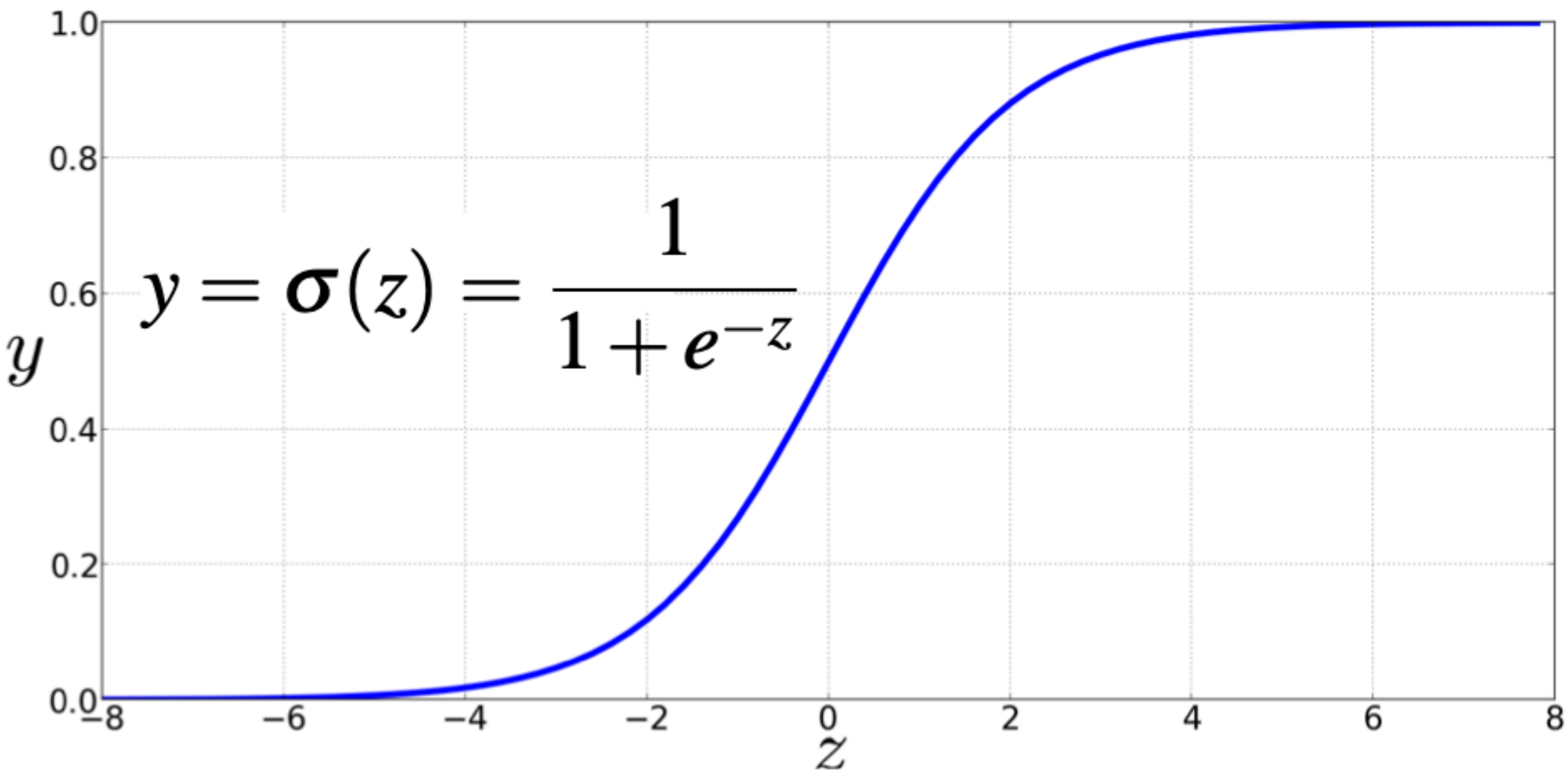
Step 2: Training Models

How does logistic regression compute predicted probabilities?

- $z = \left(\sum_{i=1}^{n} w_i x_i\right) + b = \vec{w} \cdot \vec{x} + \text{b}$

Step 2: Training Models

Sigmoid function



Step 2: Training Models

How does logistic regression compute predicted probabilities?

- $z = (\sum_{i=1}^{n} w_i x_i) + b = \vec{w} \cdot \vec{x}$ + b

- σ(z) = $\frac{1}{1+e^{-z}}$ = $\frac{1}{1+exp(-z)}$

Step 2: Training Models

How does logistic regression compute predicted probabilities?

- $p(y = 1|x) = \sigma(\vec{w} \cdot \vec{x} + b)$

- $p(y = 0|x) = 1 - \sigma(\vec{w} \cdot \vec{x} + b)$

Step 2: Training Models

How do predicted probabilities turn into binary labels ($\hat{y}$)?

$$\begin{cases} 1 & \text{if } P(y = 1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Step 2: Training Models

E.g., sentiment classification from movie reviews

"It's hokey. There are virtually no surprises, and the writing is second-rate. So why was it so enjoyable? For one thing, the cast is great, Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing? It sucked me in, and it'll do the same to you."

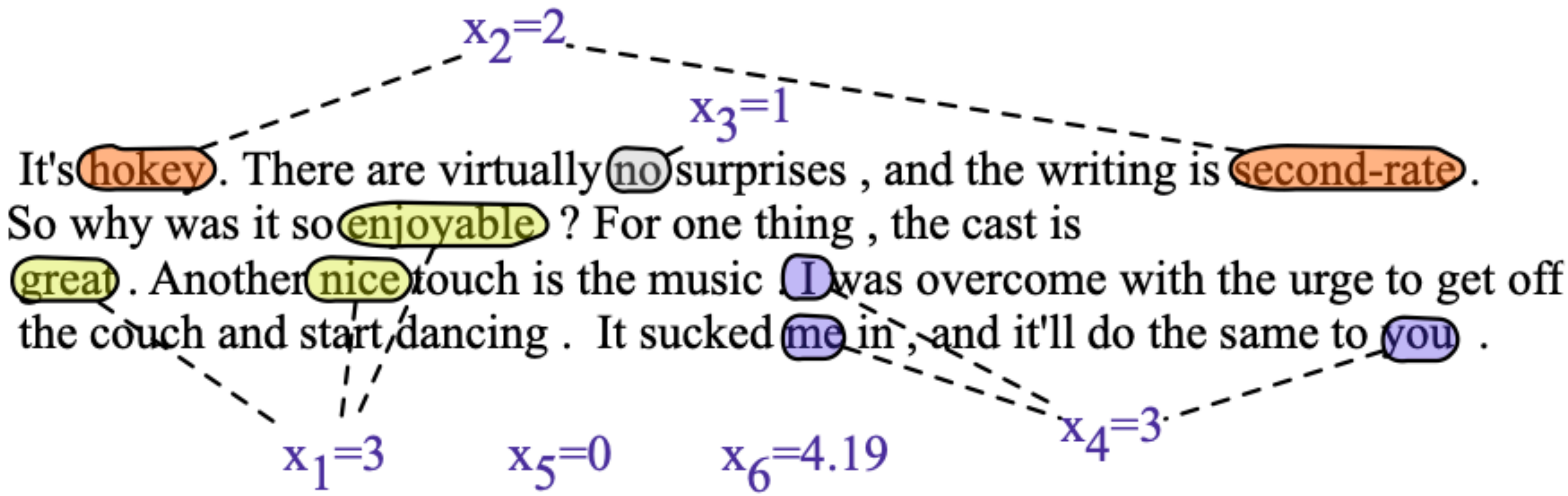
Step 2: Training Models

Sentiment classification from movie reviews

Var	Definition
x_1	count(positive lexicon) $\in$ doc)
x_2	count(negative lexicon) $\in$ doc)
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Step 2: Training Models

Sentiment classification from movie reviews



Step 2: Training Models

How are the parameters, weights $\vec{w}$ and bias b, learned?

- Goal: learn $\vec{w}$ and b that make $\hat{y}^i_{train}$ for each training observation as "close" as possible to y^i_{train} (the true label)
- Two components for estimation
  - Metric for "closeness": loss/cost function (e.g., cross entropy loss)
  - Optimization algorithm: (stochastic) gradient descent

Step 2: Training Models

Learn parameters that maximize the chance of getting the correct label (Conditional Maximum Likelihood Estimation)

- p($y|x$) = $\hat{y}^y$ * $(1 - \hat{y})^{1-y}$
- If y = 1, p($y|x$) = $\hat{y}$
  - The higher $\hat{y}$ is, the better the classifier
- If y = 0, p($y|x$) = $(1 - \hat{y})$
  - The higher $(1 - \hat{y})$ is (the lower $\hat{y}$ is), the better the classifier

Step 2: Training Models

From p($y|x$), we drive loss (cost) function

- p($y|x$) = $\hat{y}^{y}$ * $(1 - \hat{y})^{1-y}$

  → Log(p($y|x$)) = Log($\hat{y}^{y}$ * $(1 - \hat{y})^{1-y}$)

  → Log(p($y|x$)) = $y$log$\hat{y}$ + (1-y)log(1-$\hat{y}$)

We can turn it into the cross entropy loss function (that should be minimized)

- L_{CE} = -[$y$log$\hat{y}$ + (1-y)log(1-$\hat{y}$)]

Step 2: Training Models

Gradient descent

- With the cross entropy loss function, we have a metric for discrepancies
- Gradient descent is an algorithm used to find the optimal set of weights (and bias) that minimizes the discrepancies, averaged over all observations

$$\rightarrow \hat{\theta} = \underset{\theta}{\text{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{\text{CE}}(f(x^{(i)}; \theta), y^{(i)})$$

Step 2: Training Models

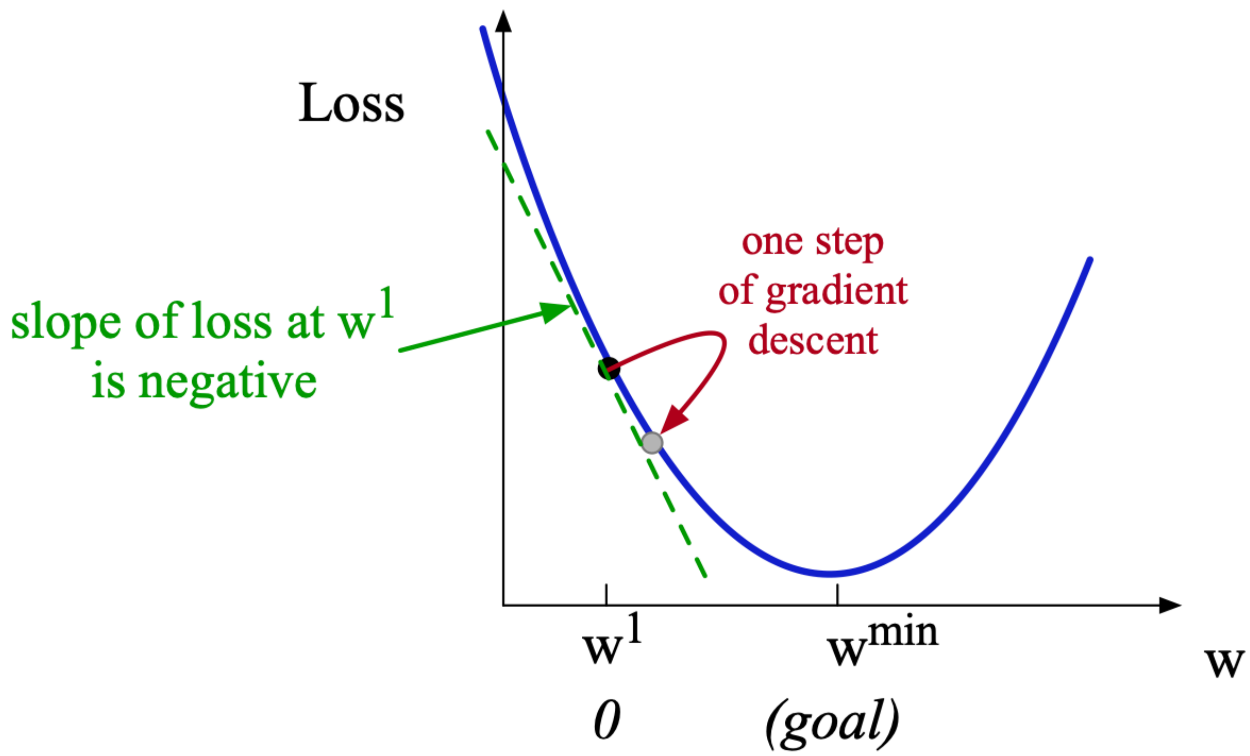
Gradient descent finds a minimum of a function

- Figures out in which direction the function’s slope rises most steeply
- Then move in the opposite direction
- Stops when it reaches the minimum

Step 2: Training Models

Gradient descent

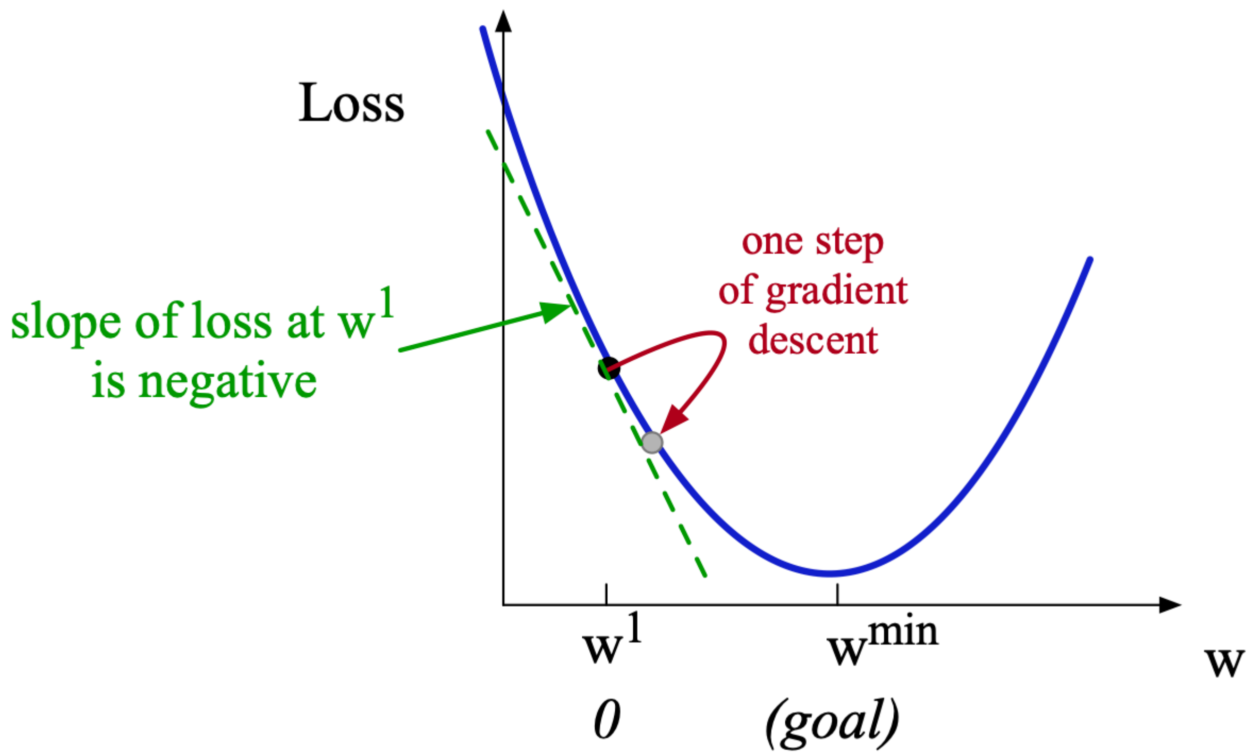
- 1-dimensional illustration



Step 2: Training Models

Gradient descent

- Updating w with learning rate (η): $w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$



Step 4: Evaluate Performance

We have

- Manually labeled documents
- Split them into C_{train} (training set) and C_{test} (test set)
- Trained a classifier on C_{train} (with y_{train} and X_{train}) $\rightarrow F(\hat{\beta^*} X)$

Now we need to evaluate its performance on C_{test}

- We compare $\hat{y}_{test}$ (predicted labels) against y_{test} (true labels)

Step 4: Evaluate Performance

Performance metrics

- Accuracy: the proportion of all predictions (both positive and negative) that the model got right
- Precision: the proportion of positive predictions that were actually correct
- Recall: the proportion of actual positives that were correctly predicted
- F-1: the harmonic (as opposed to arithmetic) mean of precision and recall

Step 4: Evaluate Performance

Confusion matrix: predictions against true labels

		True condition	
		Positive	Negative
Prediction	Positive	True Positive	False Positive (Type I error)
	Negative	False Negative (Type II error)	True Negative

Step 4: Evaluate Performance

Accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$

		True condition	
		Positive	Negative
Prediction	Positive	True Positive	False Positive (Type I error)
	Negative	False Negative (Type II error)	True Negative

Step 4: Evaluate Performance

Precision: $\frac{TP}{TP+FP}$

		True condition	
		Positive	Negative
Prediction	Positive	True Positive	False Positive (Type I error)
	Negative	False Negative (Type II error)	True Negative

Step 4: Evaluate Performance

Recall: $\frac{TP}{TP+FN}$

		True condition	
		Positive	Negative
Prediction	Positive	True Positive	False Positive (Type I error)
	Negative	False Negative (Type II error)	True Negative

Step 4: Evaluate Performance

F-1: $(2 \times precision \times recall)$ / $(precision + recall)$

Step 4: Evaluate Performance

Reminders

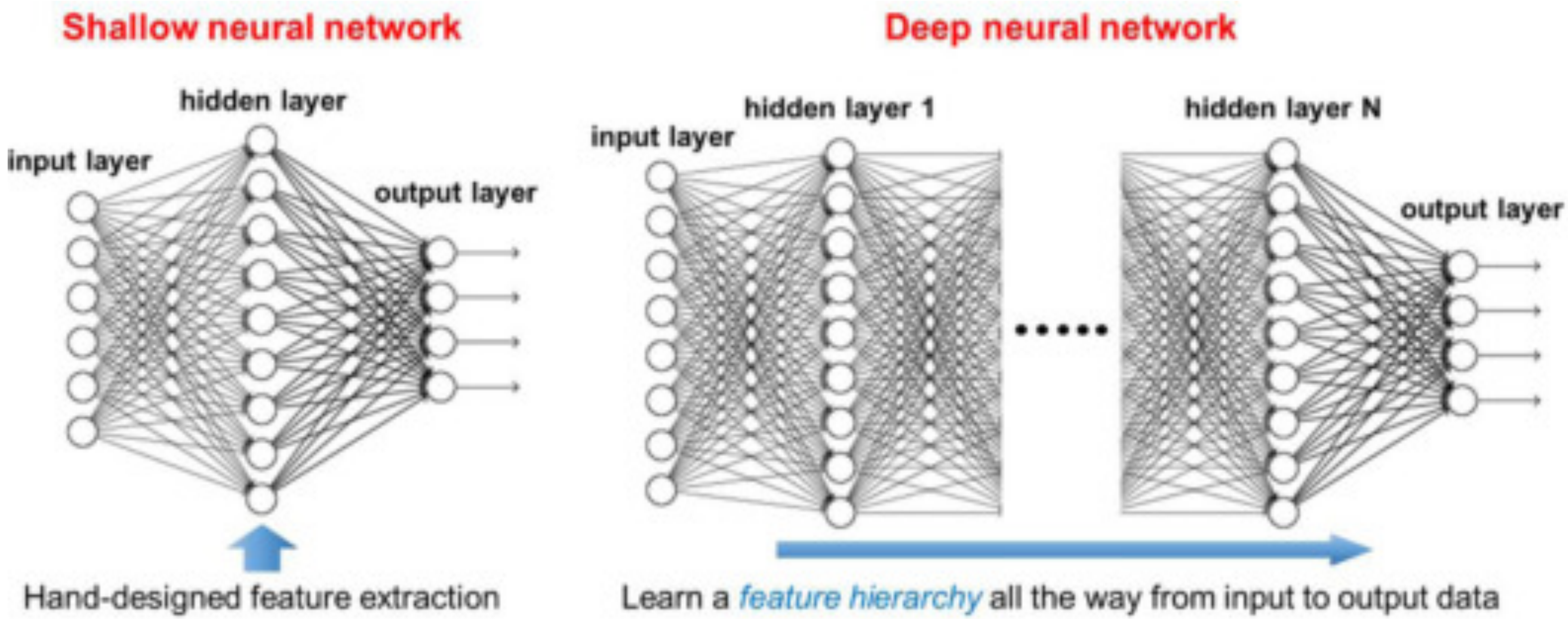
- Random train-test split: C_{train}, C_{test}
  - E.g., 10,000 comments labeled for hate speech into 8,000 and 1,000
- Our classifier learned **parameters**, maximizing performance on C_{train} and evaluating it on C_{test}

Step 4: Evaluate Performance

Parameters vs. hyper-parameters

- Parameter
  - Learned (estimated) from data (internal to the model)
  - E.g., logistic regression weights/coefficients (= β)
- Hyper-parameters
  - Defines the model structure itself (not internal to the model)
  - E.g., the size of a regularization term in logistic regression, the number of layers or learning rate in neural networks, etc.

Step 4: Evaluate Performance



Step 4: Evaluate Performance

Hyper-parameters influence model performance, and we want to “tune” them

- With different hyper-parameter values, we could fit a model configured with each value on C_{train} and evaluate performance on C_{test}
- E.g., regularization strength λ in logistic regression
  - Train different models with different λ values on C_{train}
  - Evaluate on C_{test}
  - Pick the best performing model

Step 4: Evaluate Performance

What could go wrong?

- In comparing different models (different hyperparaemter values), we might overfit on C_{train}
- By repeatedly using the specific train-test split, our comparison can be affected by the specific characteristics of the split

Step 4: Evaluate Performance

Validation set

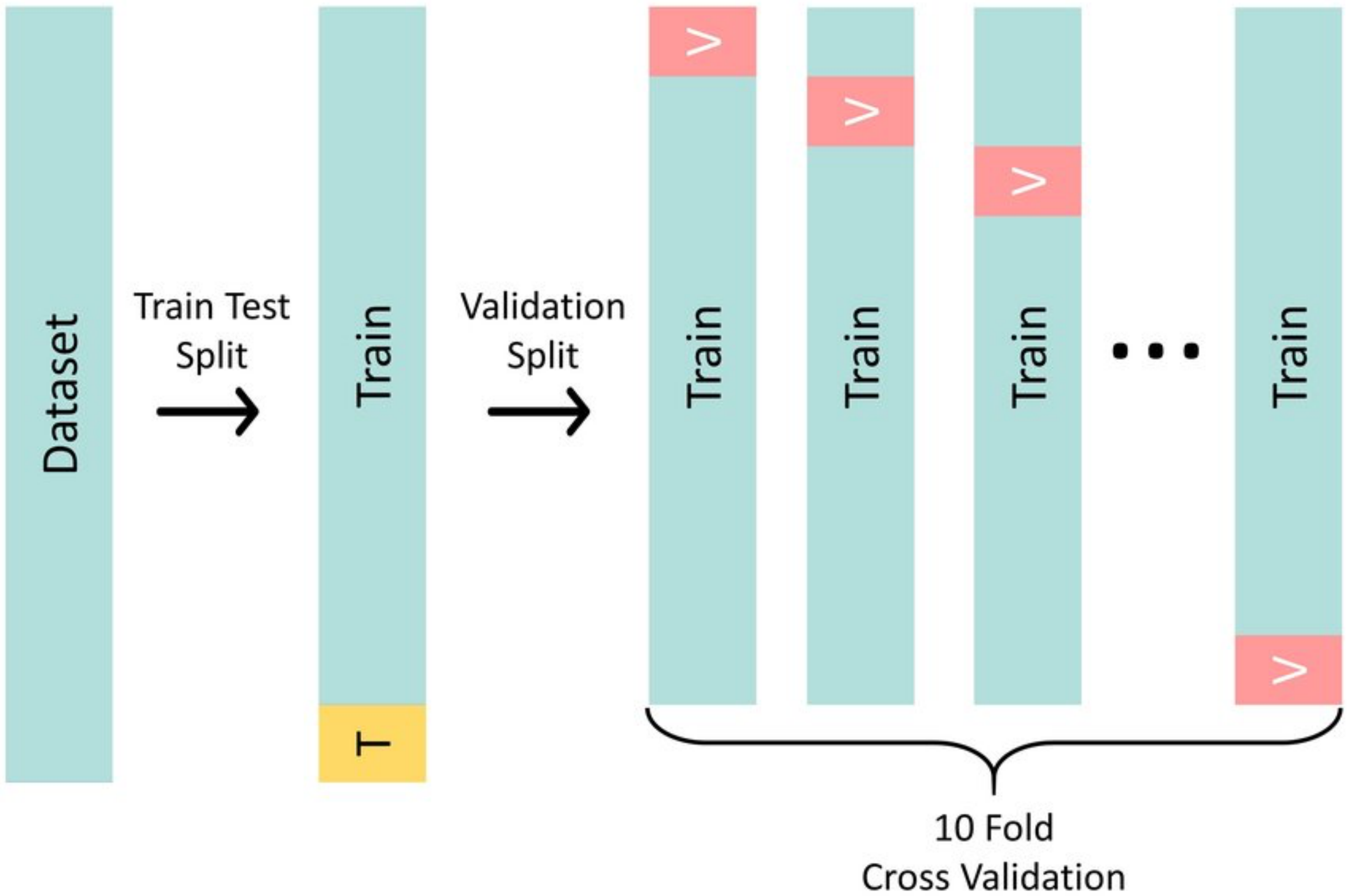
- We split the labeled data set into a training set, a “validation set”, and a test set: C_{train}, $C_{validation}$, and C_{test}
- Train a model on C_{train} and see how it performs on $C_{validation}$
- Repeat this step for multiple hyper-parameter configurations
- Pick the best-performing configuration
- Train a final model based on the configuration using C_{train} + $C_{validation}$
- Evaluate on C_{test}

Step 4: Evaluate Performance

K-fold cross-validation

- Randomly split C_{train} into K equal parts or "folds" (commony 5 or 10)
- For each iteration
  - Treat one fold as the "validation set"
  - Train your model on the remaining $K - 1$ folds
  - Evaluate performance on the validation set kept aside
- After cyclig through all iterations
  - Aggregate the performance metrics obtained from each iteration
  - Choose the classifier with the highest cross-validated performance
  - This step may involve not just hyper-parameter tuning but also things like feature repesentation, etc.
- (Re)train the chosen best classifier on C_{train} (all K folds combined) and evalute on C_{test}

Step 4: Evaluate Performance



Summary

Supervised text classification provides a highly useful tool to assign labels to texts

- Be aware of the principles of building a labeled data set
  - Conceptualization, inter-coder reliability, annotator bias, etc.
- Validate, validate, and validate!
- Choose appropriate evaluation metrics

Guided Coding

Text classification with movie reviews data (link)