CSE 156 | Lecture 2: Text Classification with Linear Models

Ndapa Nakashole

October 3, 2024

Administrative matters

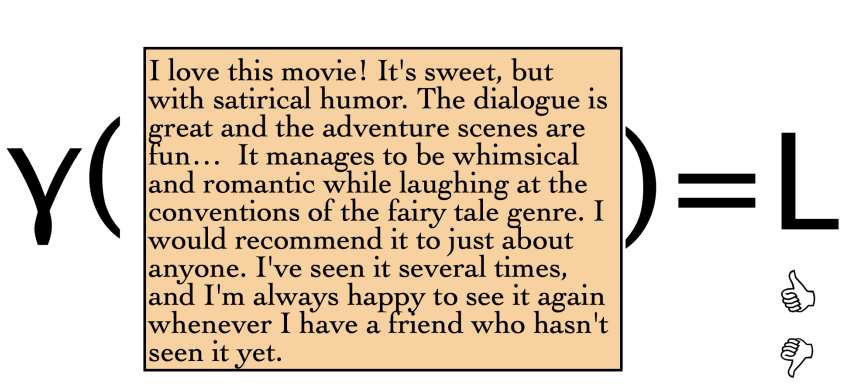
- **PA1**: out later today, or early tomorrow (2 weeks to complete)

Today

1. Text Classification: Problem Definition
2. Basic data-driven approaches to text classification: k-Nearest Neighbor, and a Linear Classifier

Text Classification

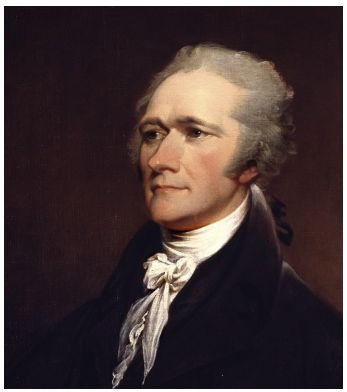
Task: assign a text document to one or more categories from a predefined set



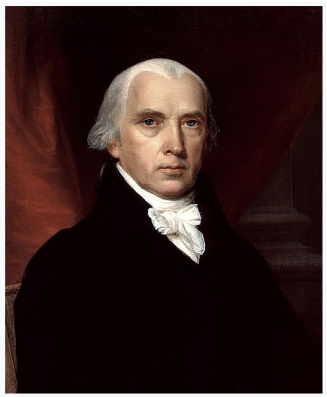
- Sentiment analysis
- Spam detection
- Topic classification
- Language identification
- Authorship attribution

Authorship Attribution: The Federalist Papers

- **85** articles published in **1787-1788** to promote the ratification of the United States Constitution

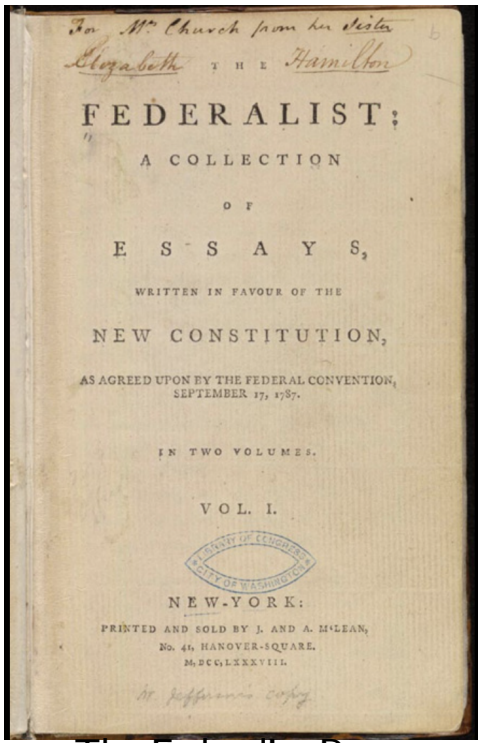


Alexander Hamilton



James Madison

John Jay



The Federalist Papers

- But who wrote which article? 12 articles were disputed
- 1963: solved by Mosteller and Wallace using Bayesian methods

Gender of the Author?

"By 1925 present-day Vietnam was divided into three parts under French colonial rule. The southern region embracing Saigon and the Mekong delta was the colony of Cochin-China; the central area with its imperial capital at Hue was the protectorate of Annam "

"Clara never failed to be astonished by the extraordinary felicity of her own name. She found it hard to trust herself to the mercy of fate, which had managed over the years to convert her greatest shame into one of her greatest asset . . . "

S. Argamon, M. Koppel, J. Fine, A. R. Shimoni, 2003. "Gender, Genre, and Writing Style in Formal Written Texts,"

Gender of author? (determiners and pronoun features)

"By 1925 present-day Vietnam was divided into three parts under French colonial rule. The southern region embracing Saigon and the Mekong delta was the colony of Cochin-China; the central area with its imperial capital at Hue was the protectorate of Annam "

"Clara never failed to be astonished by the extraordinary felicity of her own name. She found it hard to trust herself to the mercy of fate, which had managed over the years to convert her greatest shame into one of her greatest asset . . . "

S. Argamon, M. Koppel, J. Fine, A. R. Shimoni, 2003. "Gender, Genre, and Writing Style in Formal Written Texts,"

Terminology around Number of Classes

- Binary classification: Two classes
- Multi-class classification: More than two classes
- Multi-label classification: Assign multiple labels to a single instance

A Cancer Conundrum: Too Many Drug Trials, Too Few Patients

Breakthroughs in immunotherapy and a rush to develop profitable new treatments have brought a crush of clinical trials scrambling for patients.

By GINA KOLATA

→ Health

Yankees and Mets Are on Opposite Tracks This Subway Series

As they meet for a four-game series, the Yankees are playing for a postseason spot, and the most the Mets can hope for is to play spoiler.

By FILIP BONDY

→ Sports

~20 classes

A Text Classifier

- Unlike tasks with clear rules such as sorting numbers . . .
- Early days of NLP tried to write rules but . . .
  - **Semantic Gap**: The computer doesn't know what words mean, they are just symbols, no knowledge of their relationships, sentiment, etc.
  - **Intra-class variations**: There are many ways to be a particular label (e.g. SPAM email)
  - **Scalability**: Have to write rules for every class label

Machine Learning: A Data-Driven Approach

- Text classification lacks a straightforward, one-size-fits-all algorithm. . . .
- Instead use Machine Learning

> *Arthur Samuel (1959): Machine learning is the field of study that gives computers the ability to* ***learn without being explicitly programmed****.*

Machine Learning: Data-Driven Approach

1. Collect a **dataset** of example text inputs and their labels
2. Use Machine Learning algorithms to **train** a classifier on the training examples
3. **Evaluate** the classifier on new text data
   - A good classifier will **generalize** well - i.e., predict unseen examples correctly

```
def train(training_texts, labels):
    # Machine learning!
    return model

def predict(model, test_text):
    # Use model to predict labels
```

Input Representation

- Need to **represent text in a way that the ML models can process**.

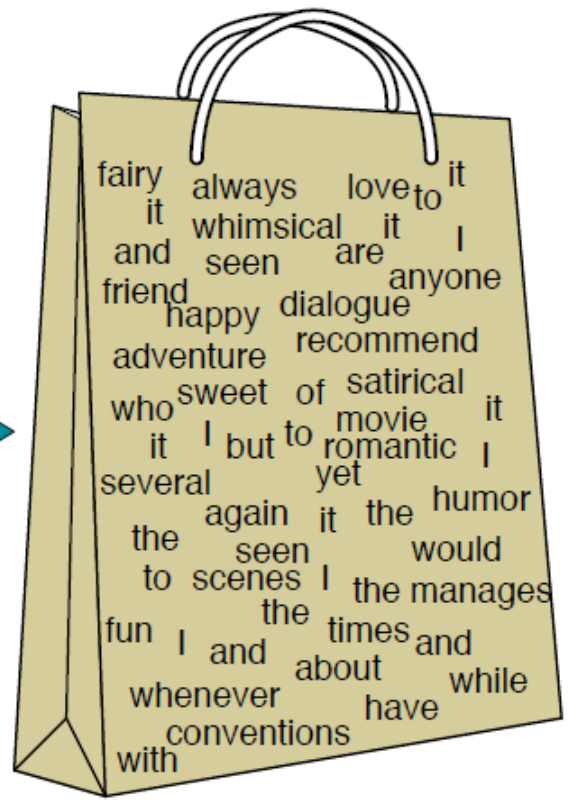
- Machine Learning models require **numerical input**
- Given some text, we need to convert it into a numerical feature vector

$$\mathbf{x} = f(text)$$

Text Representations with Bag of Words (BoW)

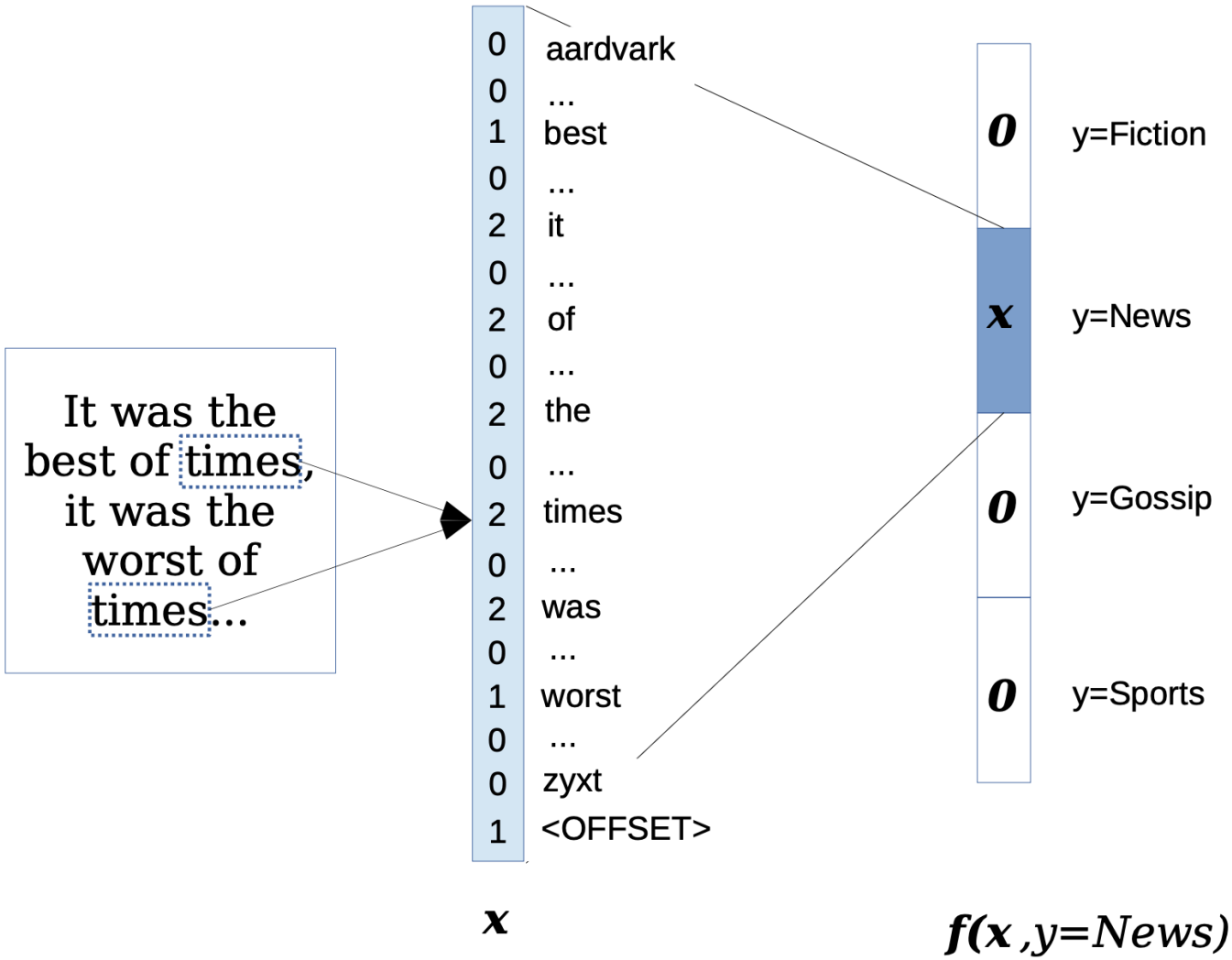
- **Bag of Words (BoW)**: Represents text as an unordered collection of words, disregarding grammar and word order

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



Feature Vector: Bow of Words

BoW produces a vector of counts from text - can now use ML!



Challenges with BOW feature vectors

- **Sparsity**: High-dimensional, sparse vectors (size of vocabulary)
- **Loss of Sentence Structure:** Sentence structure is lost. These two sentences have the same representation
  - The movie was not good, it was bad
  - The movie was not bad, it was good
- **Semantic Gap:** BoW representation lacks word meaning
  - e.g., no idea that *good* and *great* have similar meanings
- Coming later - a more powerful representation - **word embeddings** to address these challenges

First classifier: # Nearest Neighbor Classifier

First classifier: Nearest Neighbor

```
def train(training_texts, labels):
    # Machine learning!
    return model
```

→ Memorize all data and labels

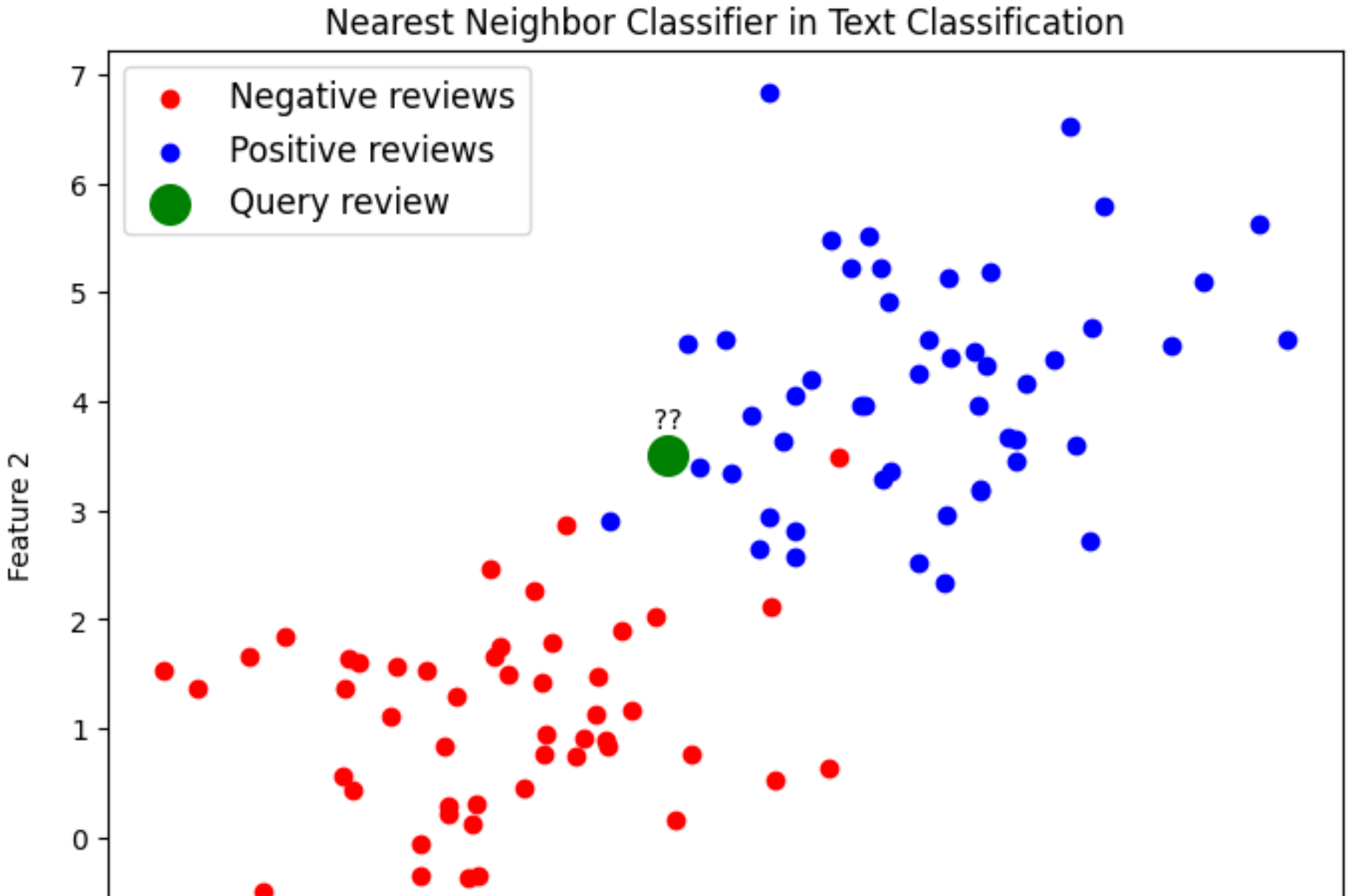
```
def predict(model, test_text):
    # Use model to predict labels
    return test_label
```

→ Predict the label of the most similar training text document

L1 distance:

$$d(x, x') = \sum_{i=1}^{n} | x_i - x'_i |$$

First classifier: Nearest Neighbor



Nearest Neighbor Classifier

```python
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype=self.ytr.dtype)
        # loop over all test rows
        for i in range(num_test):
            # find the nearest training document to the i'th test document
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis=1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example
        return Ypred
```

Nearest Neighbor Classifier

```python
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype=self.ytr.dtype)
        # loop over all test rows
        for i in range(num_test):
            # find the nearest training document to the i'th test document
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis=1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example
        return Ypred
```

Memorize the training data

Nearest Neighbor Classifier

```python
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype=self.ytr.dtype)
        # loop over all test rows
        for i in range(num_test):
            # find the nearest training document to the i'th test document
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis=1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example
        return Ypred
```

For each test document:
Find closest train document, predict label of nearest document

L1 Distance Computation

```
distances = np.sum(np.abs(self.Xtr - X[i, :]), axis=1)
```

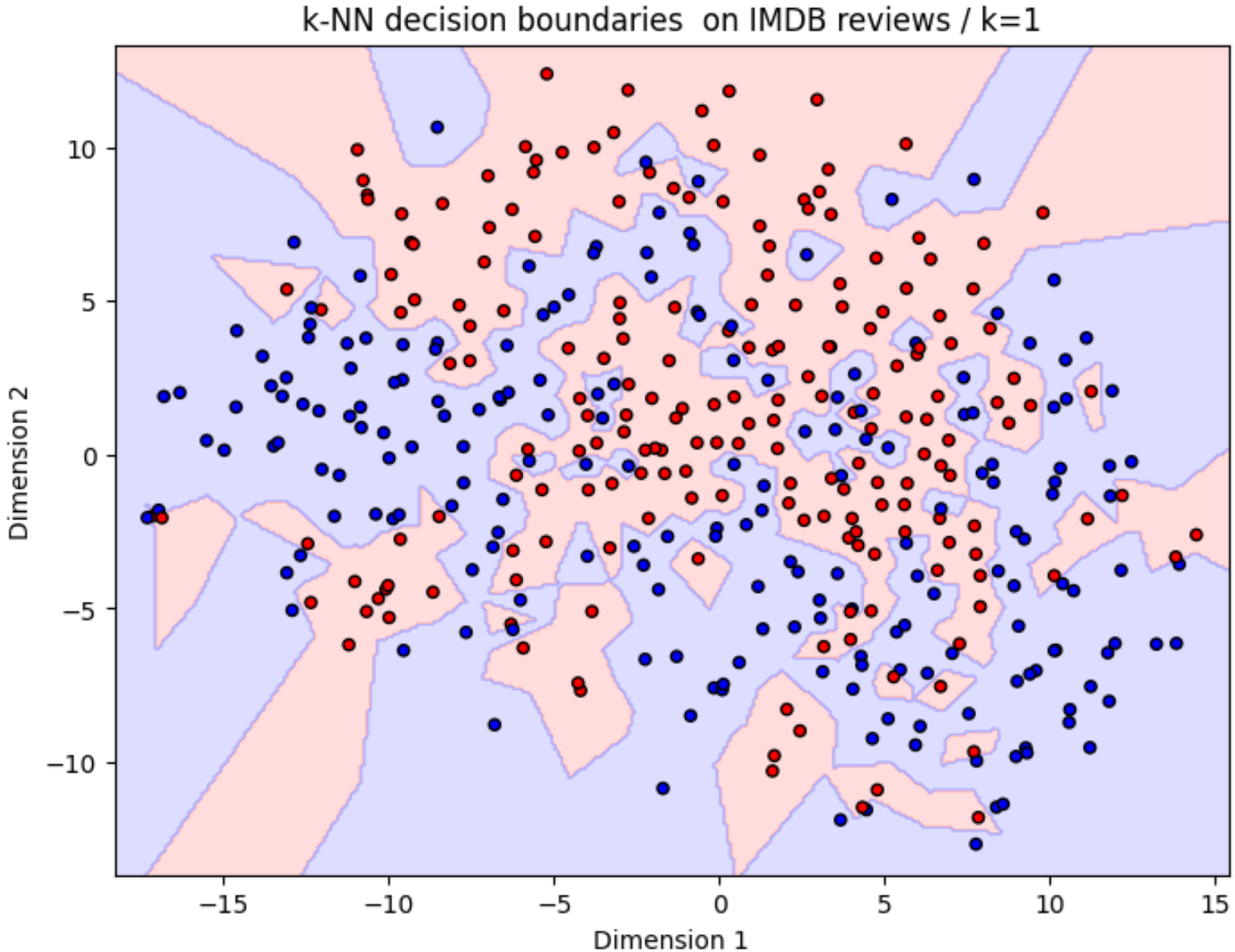
- `self.Xtr` - training examples $N \times D$,
- `X[i, :]` is the i-th test example $1 \times D$
- `self.Xtr - X[i, :]` subtracts the i-th test point from each training example, giving an $N \times D$ matrix of differences
- `np.abs(self.Xtr - X[i, :])` computes the absolute value of these differences element-wise
- `np.sum(..., axis=1)` sums the absolute differences along each row (i.e., across features), resulting in a 1D array of length N, where each value is the L1 distance between the i-th test example and a training example.

Nearest Neighbor Classifier

```python
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype=self.ytr.dtype)
        # loop over all test rows
        for i in range(num_test):
            # find the nearest training document to the i'th test document
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis=1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example
        return Ypred
```

- Q: How do training and prediction time scale with the number of training examples N?
- Ans: Train O(1), predict $\mathrm{O(N)}$
- This is bad: we want classifiers that are **fast** at prediction; **slow** for training is ok

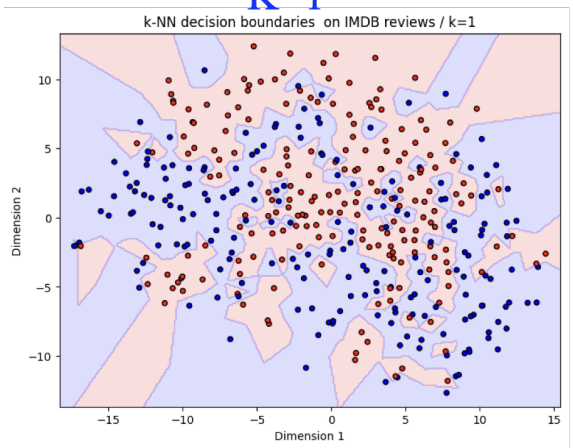
Nearest Neighbor classifier: What does the decision boundary look like?



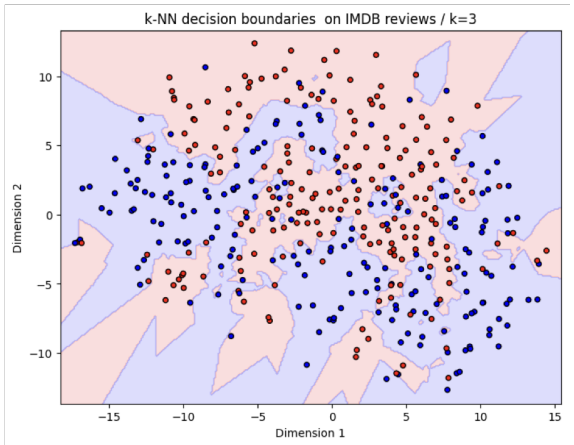
k-Nearest Neighbors: 1-NN is Unstable, use Majority Vote

Instead of copying label from nearest neighbor, take **majority vote** from k closest points

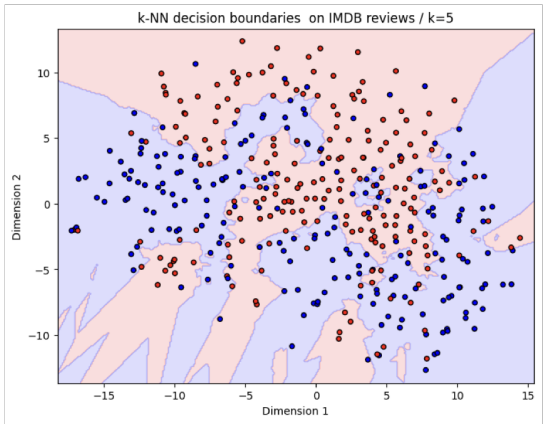
K=1



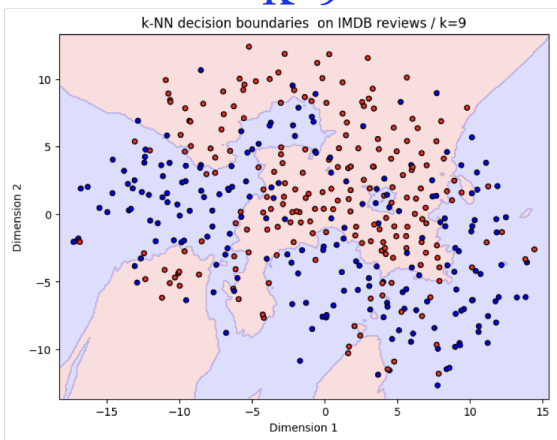
K=3



K=5



K=9

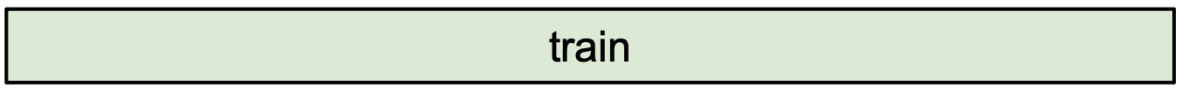


Hyperparameters

- What is the best value of $\mathbf{k}$ to use?
- What is the best distance metric to use?
- What is the best representation to use (Bow, Word Embeddings, ...)
  - These are **hyperparameters**: external settings of the model
  - **Very problem/dataset-dependent**. Must try them all out and see what works best.

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the **training data**



BAD: $K = 1$ always works perfectly if we use training data as test data

Idea #2: Choose hyperparameters that work best on the **test data**

BAD: No idea how algorithm will perform on new data

Idea #3: Split data into **train, validation**; choose **hyperparameters on validation** and **evaluate on test**

Better!

k-NN: Summary

- The **k-NN classifier** predicts labels based on the k nearest **training examples**
  - Distance metric, k, representation, are **hyperparameters**
  - Choose hyperparameters using the **validation set**
- Golden rule: Only run on the **test set once** at the very end!

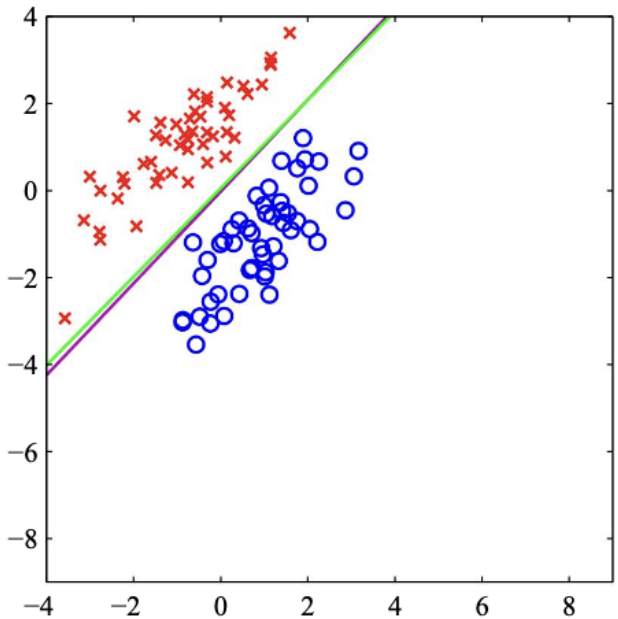
k-NN is a non-parametric method

- k-NN makes no assumptions about the underlying data distribution
  - Assumes **no specific form** for the function that maps inputs to outputs
  - Decision boundary can take on **arbitrary** shapes and complexity as k or data changes

Second classifier: Linear Classifier

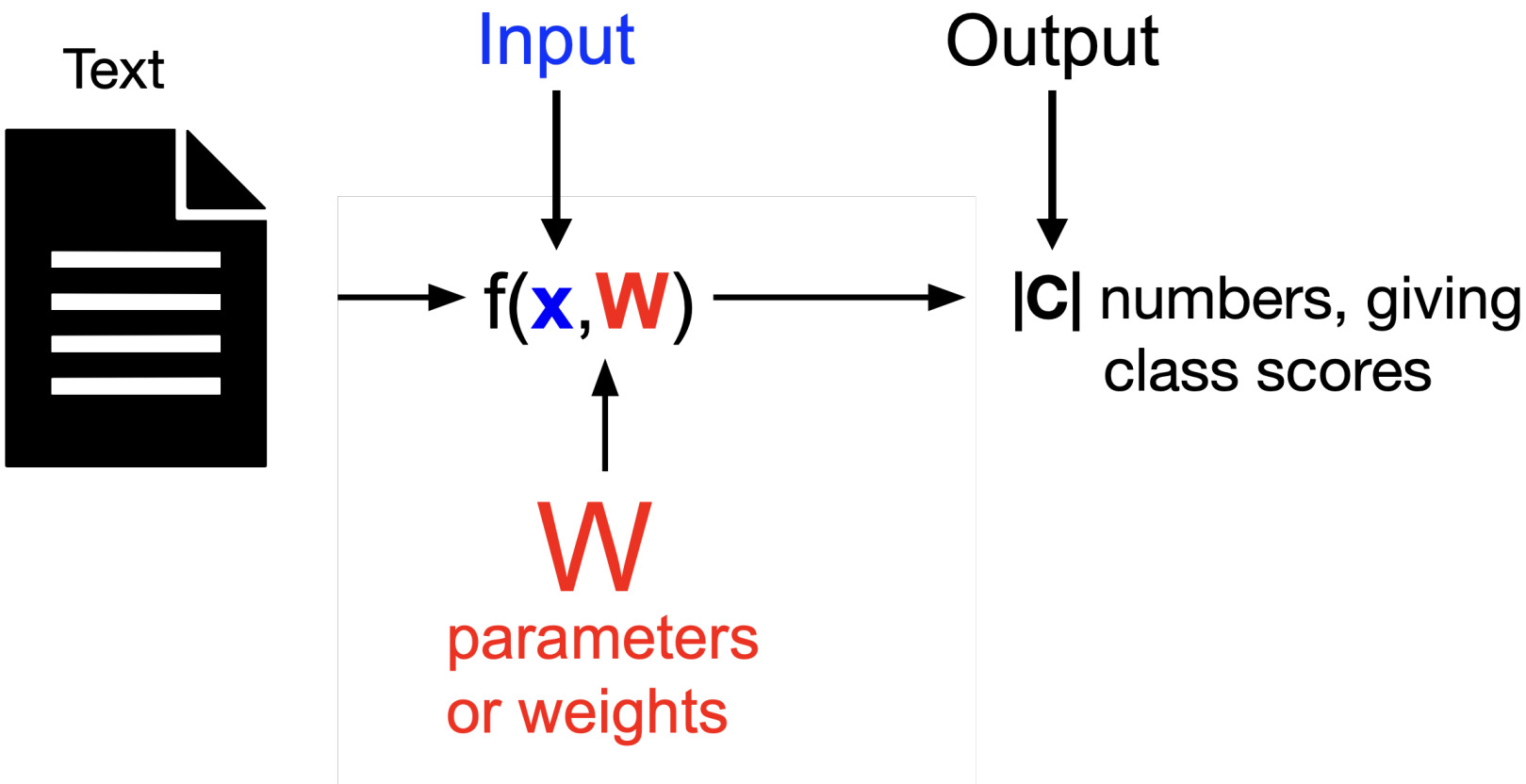
Linear Classifier: a Parametric Approach

- **Predefined Model Complexity**: assume a specific form for the decision boundary
  - straight line
  - defined by a set of fixed set of **parameters**: intercept term b and slope w



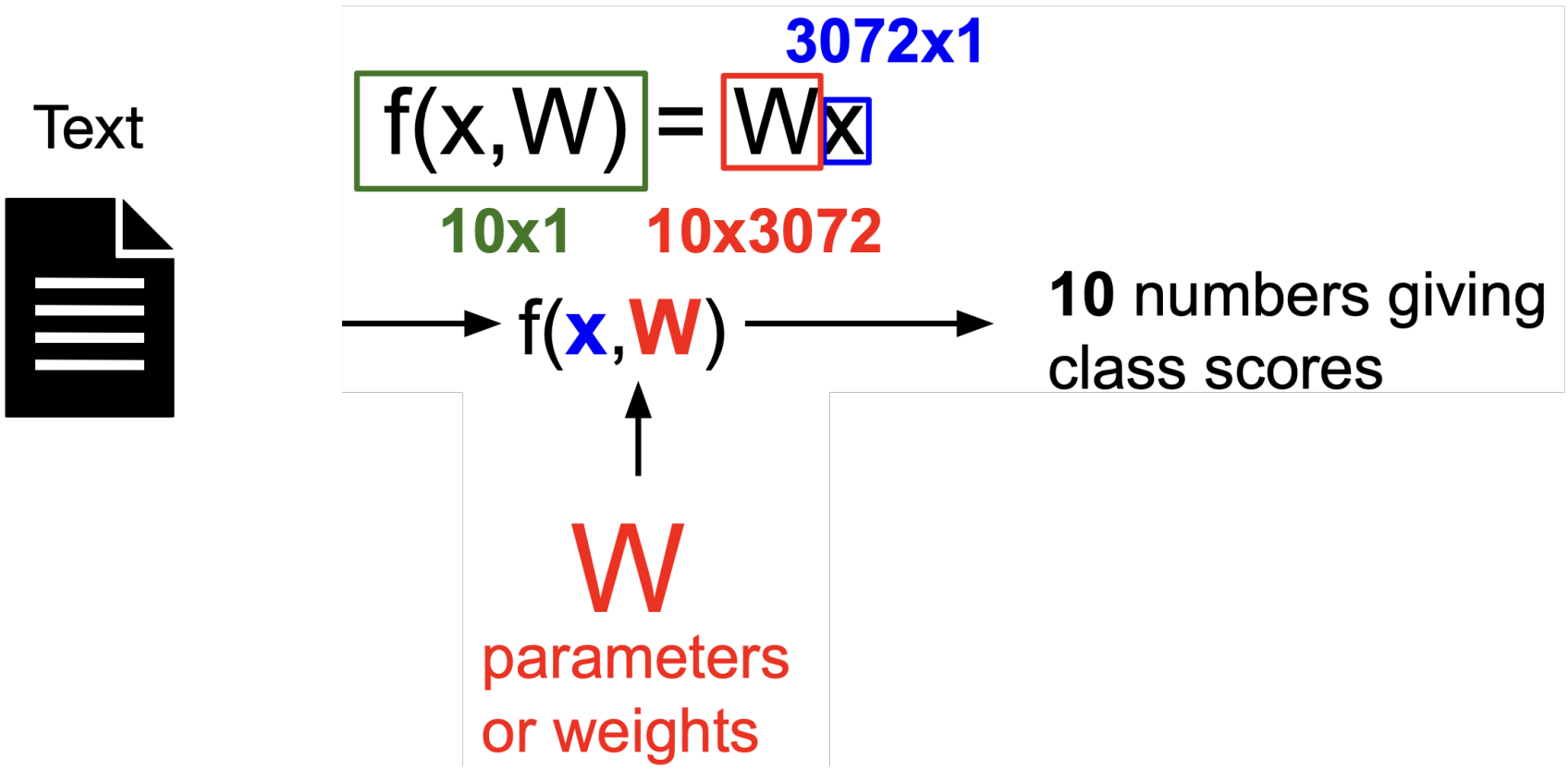
Linear Classifier

A function f of the input features x, and the parameters W:



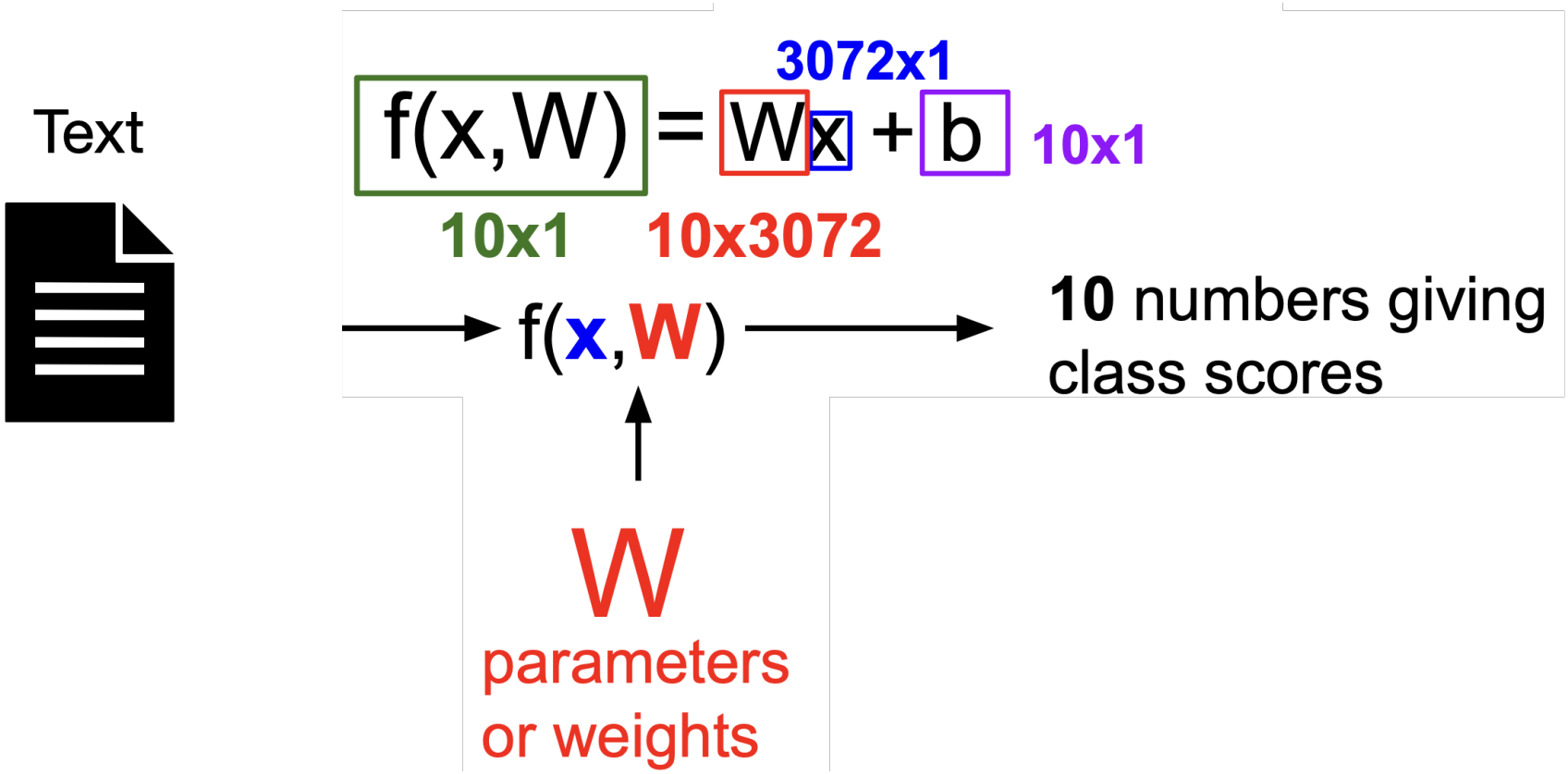
Linear Classifier

- Consider a 10 way classification problem
- 10 classes, 3072 features
- $f(x, W) = Wx$



Linear Classifier

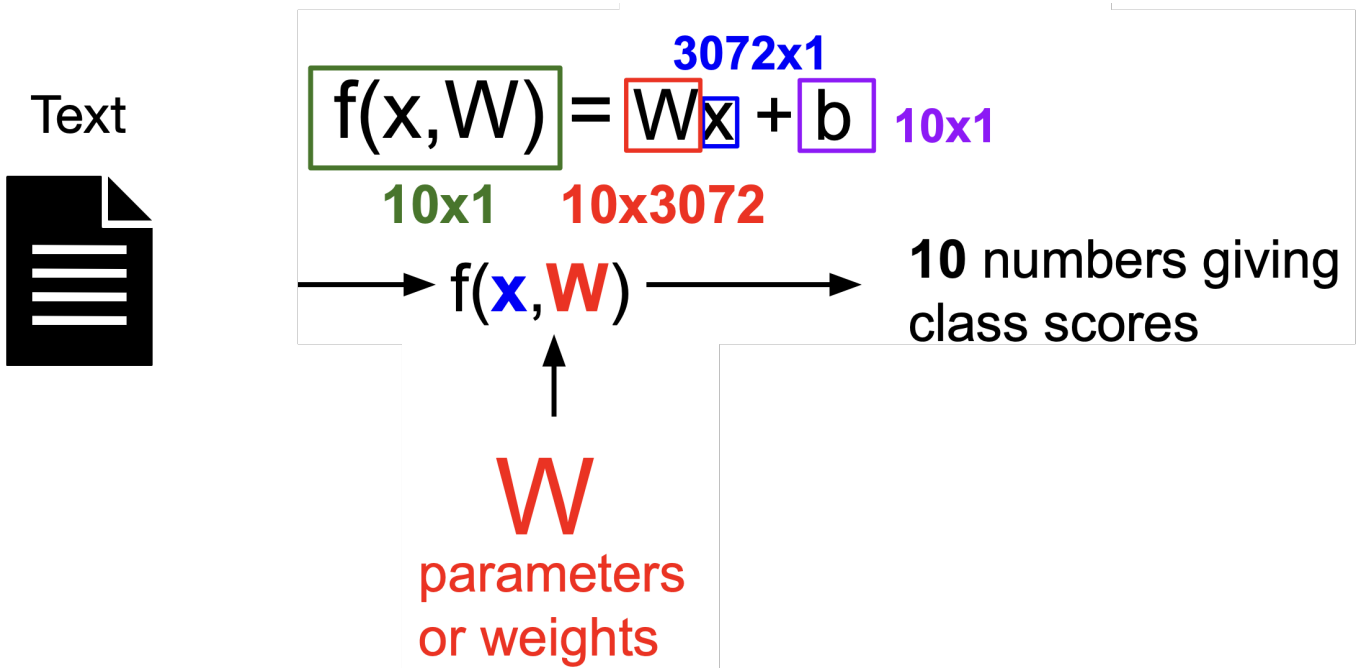
- Consider a 10 way classification problem
- 10 classes, 3072 features (assume this is the vocabulary size)
- Bias term



Bias term

- Bias term helps incorporate prior knowledge
  - consider spam vs non-spam email classification
  - prior knowledge: 90% of emails are non-spam
  - The model should predict non-spam, unless the content of the email suggests otherwise

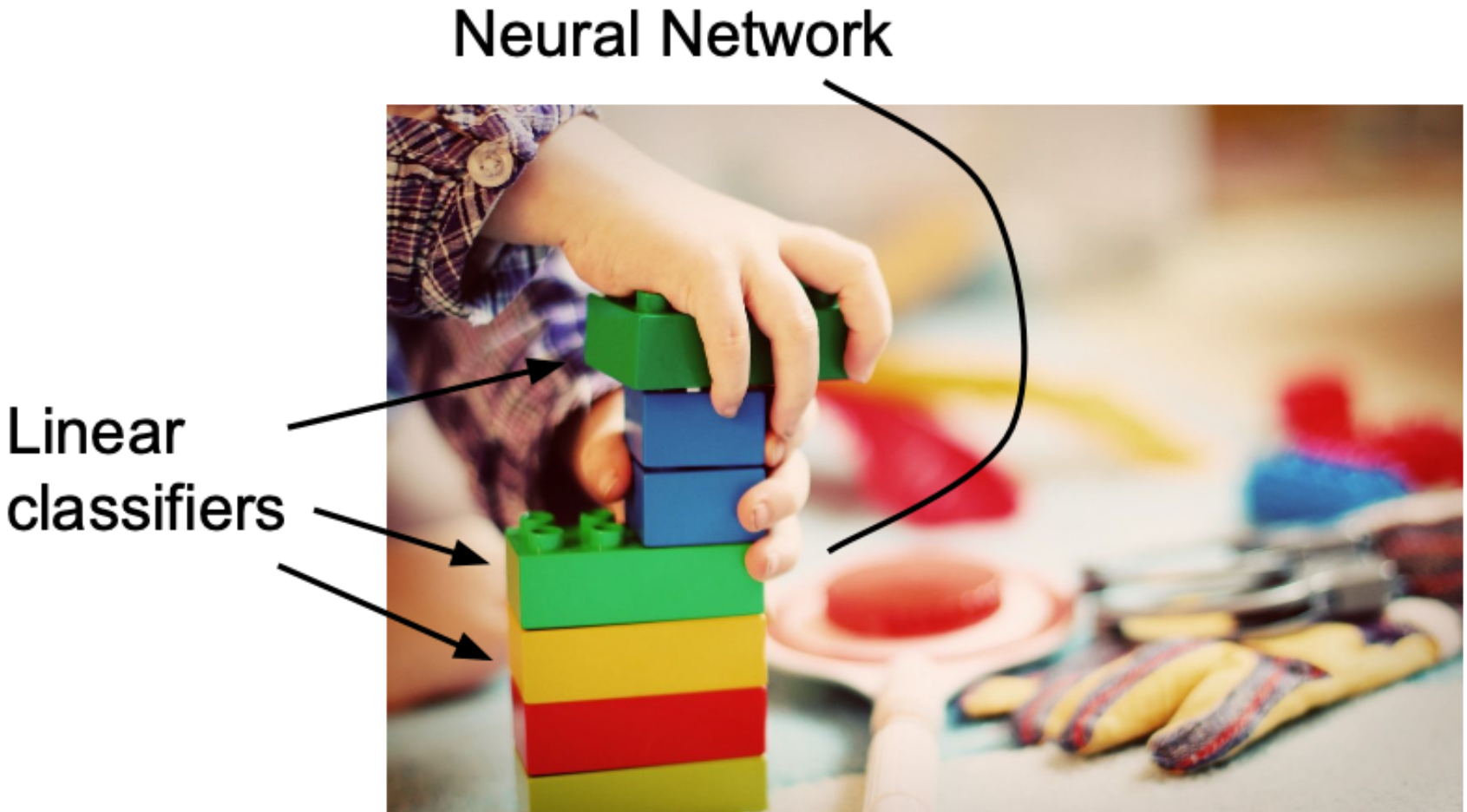
Question: Linear Classifier on more examples



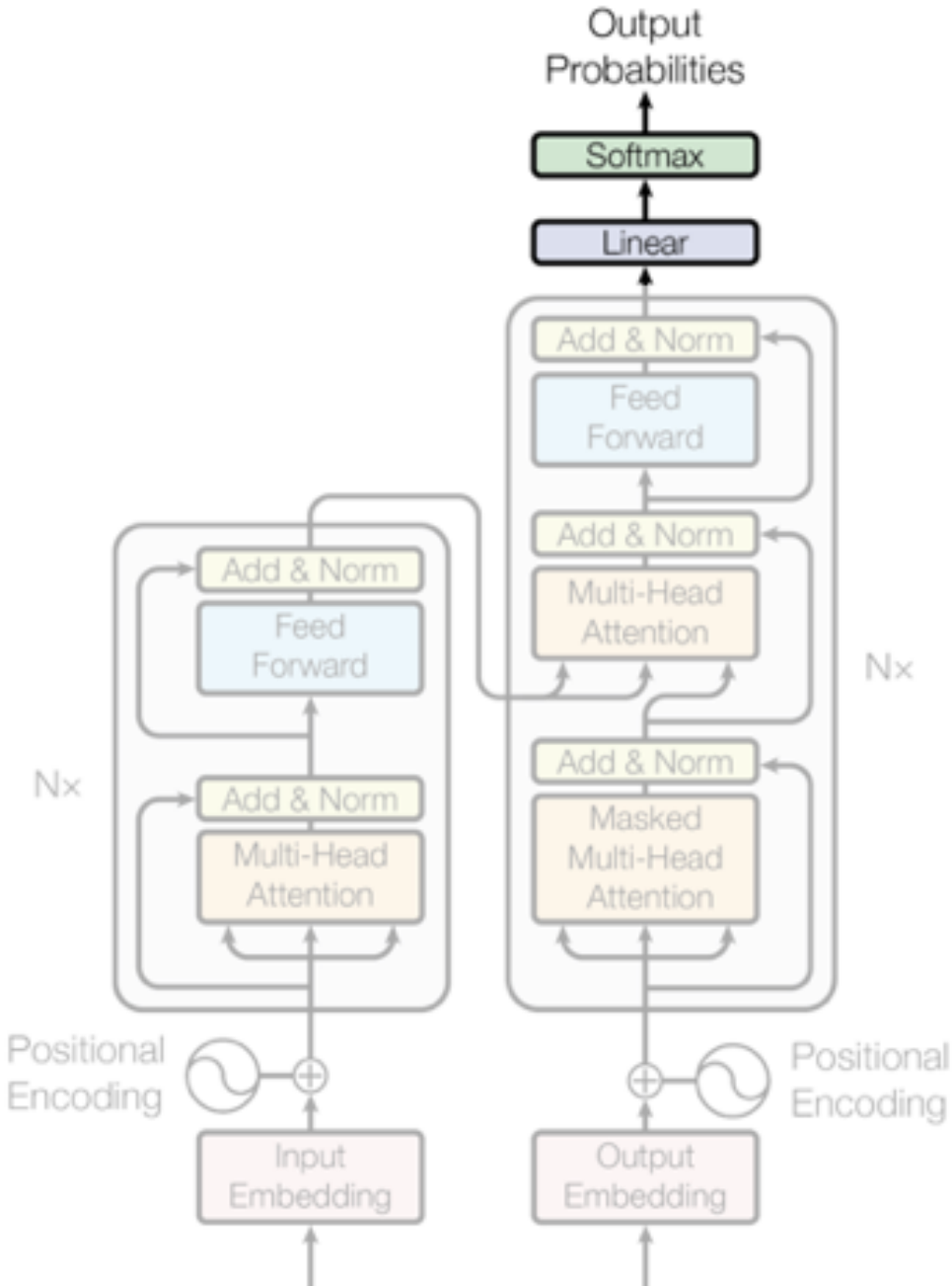
. That is, for each of the 50 examples, we have a distribution over the 10 classes.

Linear Layers: a Building Block in Deep Learning

- Neural Networks: consist of small functions, like lego blocks, that are easily stacked together to form a complex function
- Linear layers: a common building block in neural networks

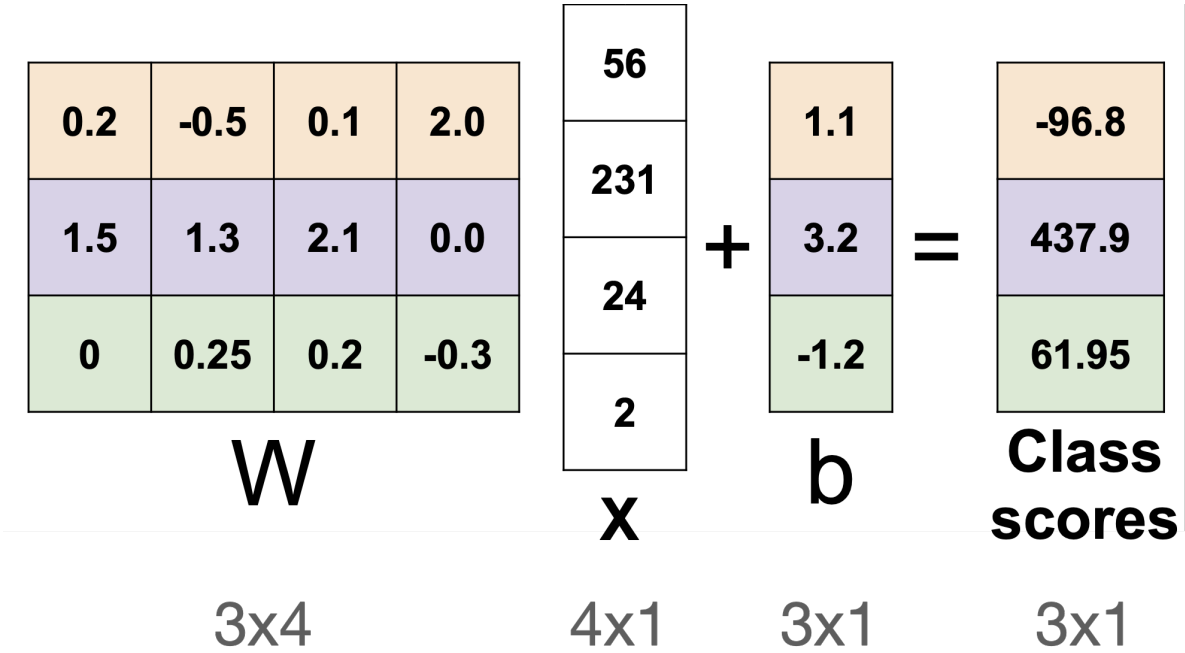


Linear Layers will keep coming back, e.g. in Transformers



Example

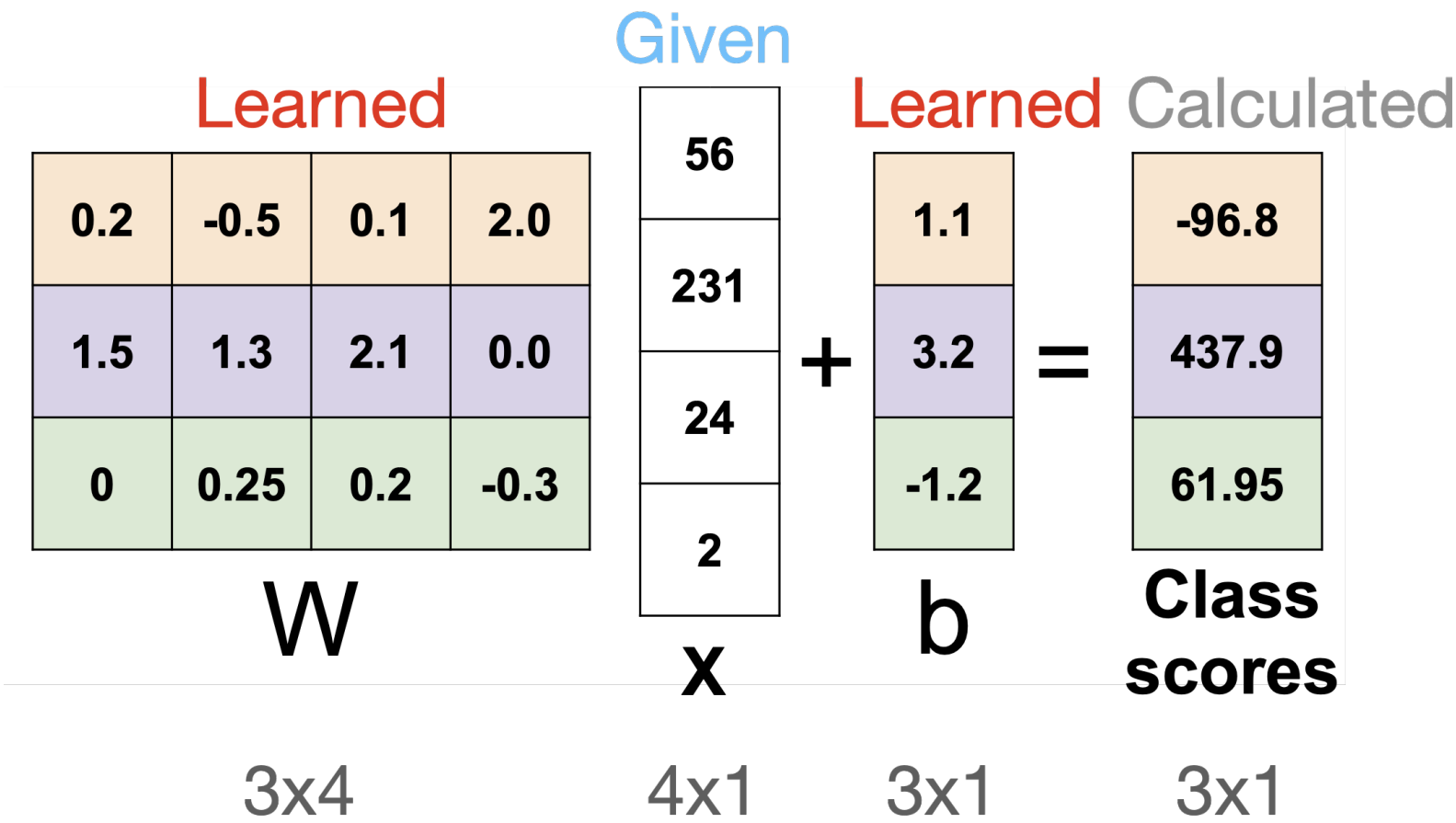
Example with a 4-dimensional email representation x, and 3 classes (Spam/Personal/Work)



first class score: $(0.2 \cdot 56) + (-0.5 \cdot 231) + (0.1 \cdot 24) + (2.0 \cdot 2) = 11.2 - 115.5 + 2.4 + 4.0 = -97.9 + 1.1 = -96.8$

One interpretation: each row of W is a template for a class. The score is the similarity between the input and the template.

Where do the numbers come from?



Linear Classifier - Choose a good W

Which W is the best? **Parameter estimation** takes as input training examples, and produces as output parameter setting $\mathbf{W}$

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data
2. Come up with a way of efficiently finding the parameters that minimize the loss function. (**optimization**)

Loss

A loss function tells us how good our current classifier is.
Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is text document and y_i is the (integer) label

Loss over the dataset is the average of loss over examples:

$$L = \frac{1}{N} \sum_{i} \underbrace{L_i \left(f\left(x_i, W\right), y_i \right)}$$

Loss for the i-th example
with $f(x_i,W)$ as the classifier prediction y_i is the gold label

Softmax Classifier (Logistic Regression)

$$\underbrace{s}_{\text{Unnormalized class scores}} = f(x_i; W) = \underbrace{Wx_i}_{\text{Weighted sum of inputs}}$$

but, we want to interpret raw classifier scores as probabilities. So we can talk about the likelihood of a document belonging to a particular class.

$$P(Y = k \mid X = x_i) = \underbrace{\frac{e^{s_k}}{\sum_j e^{s_j}}}_{\text{Softmax function}}$$

Example email

Subject: Unlock Your Inheritance Now!

Dear Esteemed Beneficiary,

We are thrilled to announce that you have been chosen to receive a substantial inheritance sum from a distant relative you never knew you had! With our easy three-step claim process, you could be enjoying your newfound wealth in no time!

To unlock your inheritance of $5,000,000.00, please provide your bank account details, social security number, and a small processing fee of $200. This offer is urgent and expires soon, so act quickly to secure your fortune!

Don't let this life-changing opportunity pass you by. Contact us immediately to begin the claim process and start living the life you deserve!

Best Regards,
Mr. John Q. Fake
Legacy Wealth Transfer Specialist

Softmax Classifier (Logistic Regression)

$$s = f(x_i; W) = Wx_i$$

Want to interpret raw classifier scores as probabilities

$$P(Y = k \mid X = x_i) = \underbrace{\frac{e^{s_k}}{\sum_j e^{s_j}}}_{\text{Softmax function}}$$

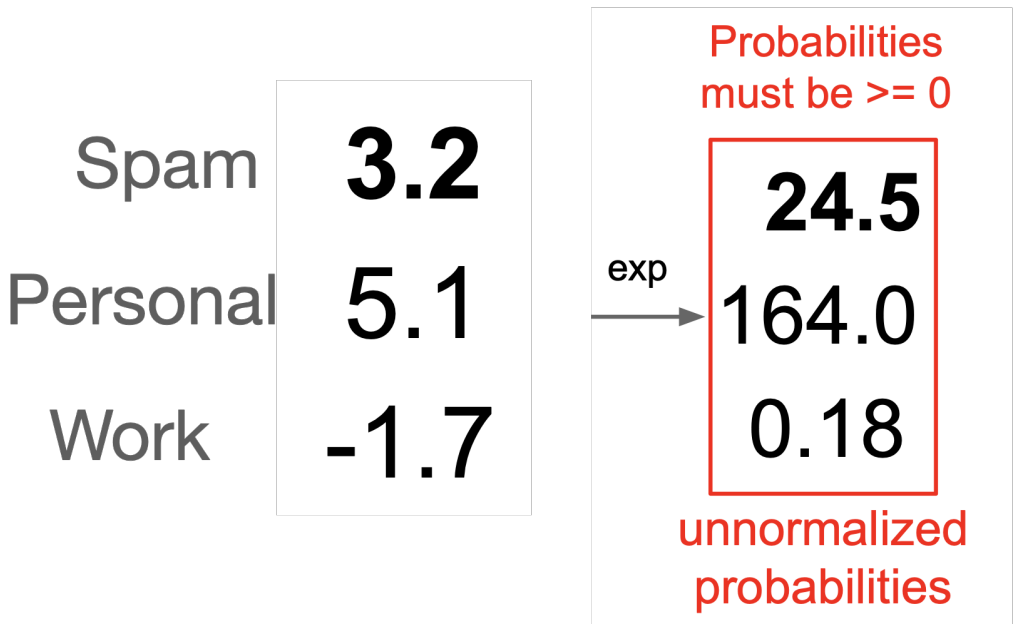
Spam	**3.2**
Personal	5.1
Work	-1.7

Softmax Classifier (Logistic Regression)

$$s = f(x_i; W) = Wx_i$$

Want to interpret raw classifier scores as probabilities

$$P(Y = k \mid X = x_i) = \underbrace{\frac{e^{s_k}}{\sum_j e^{s_j}}}_{\text{Softmax function}}$$

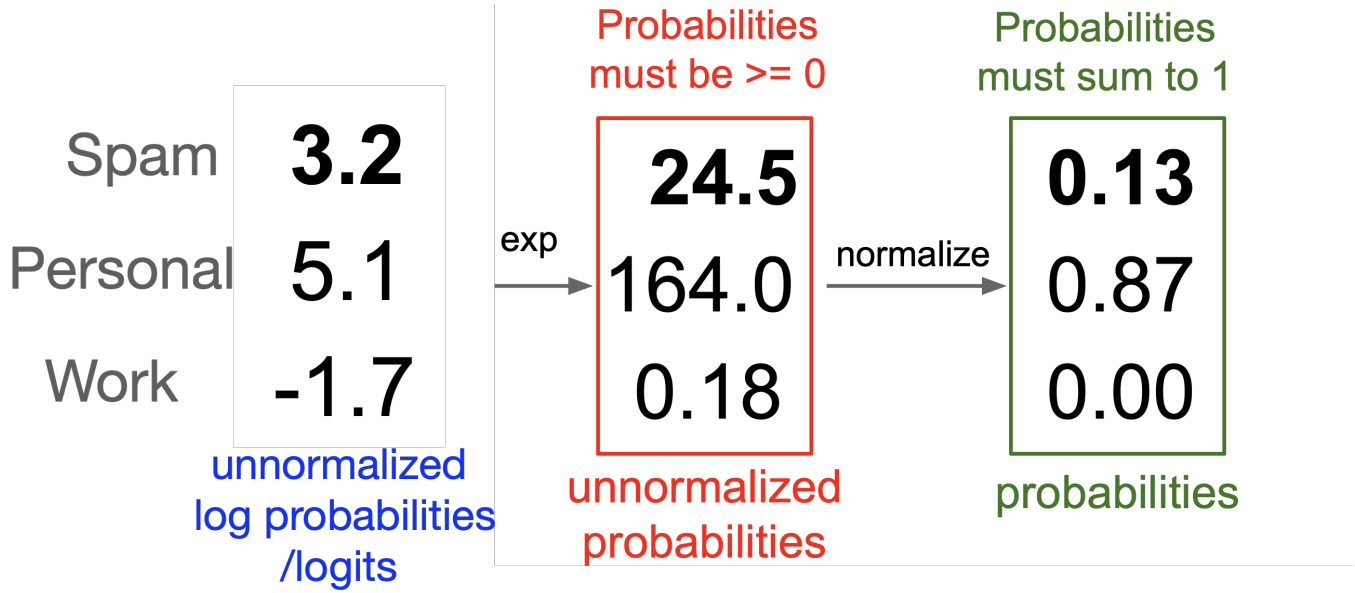


Softmax Classifier (Logistic Regression)

$$s = f(x_i; W) = Wx_i$$

Want to interpret raw classifier scores as probabilities

$$P(Y = k \mid X = x_i) = \underbrace{\frac{e^{s_k}}{\sum_j e^{s_j}}}_{\text{Softmax function}}$$

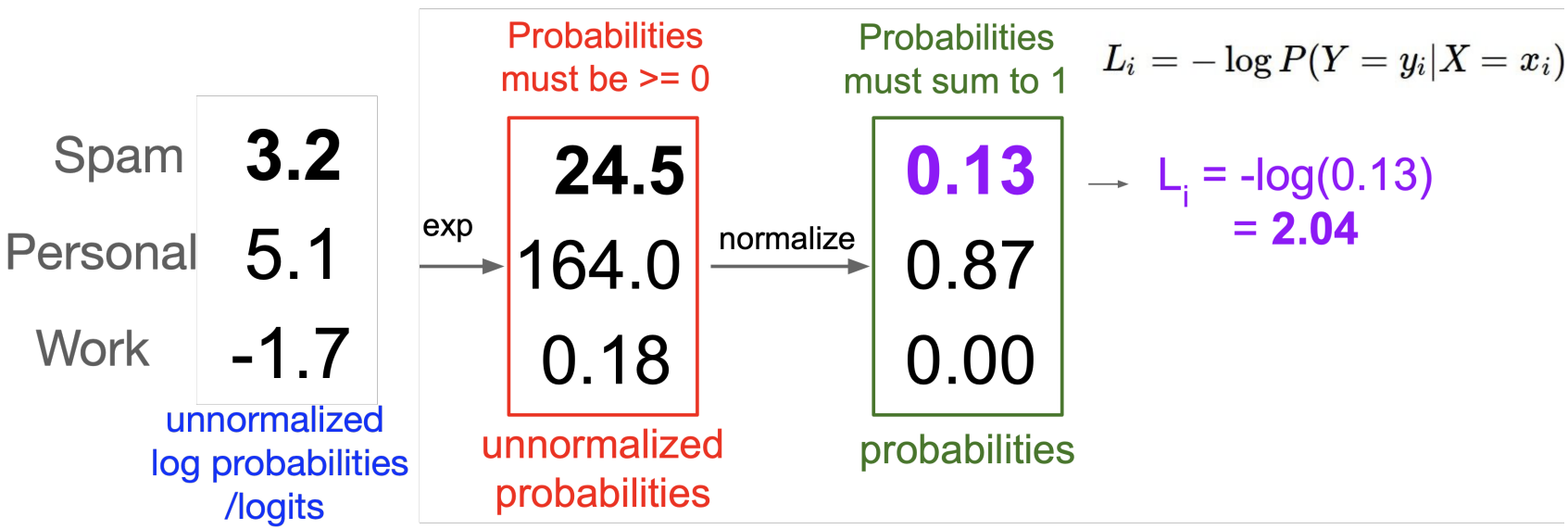


Softmax Classifier (Logistic Regression)

$$s = f(x_i; W) = Wx_i$$

Want to interpret raw classifier scores as probabilities

$$P(Y = k \mid X = x_i) = \underbrace{\frac{e^{s_k}}{\sum_j e^{s_j}}}_{\text{Softmax function}}$$

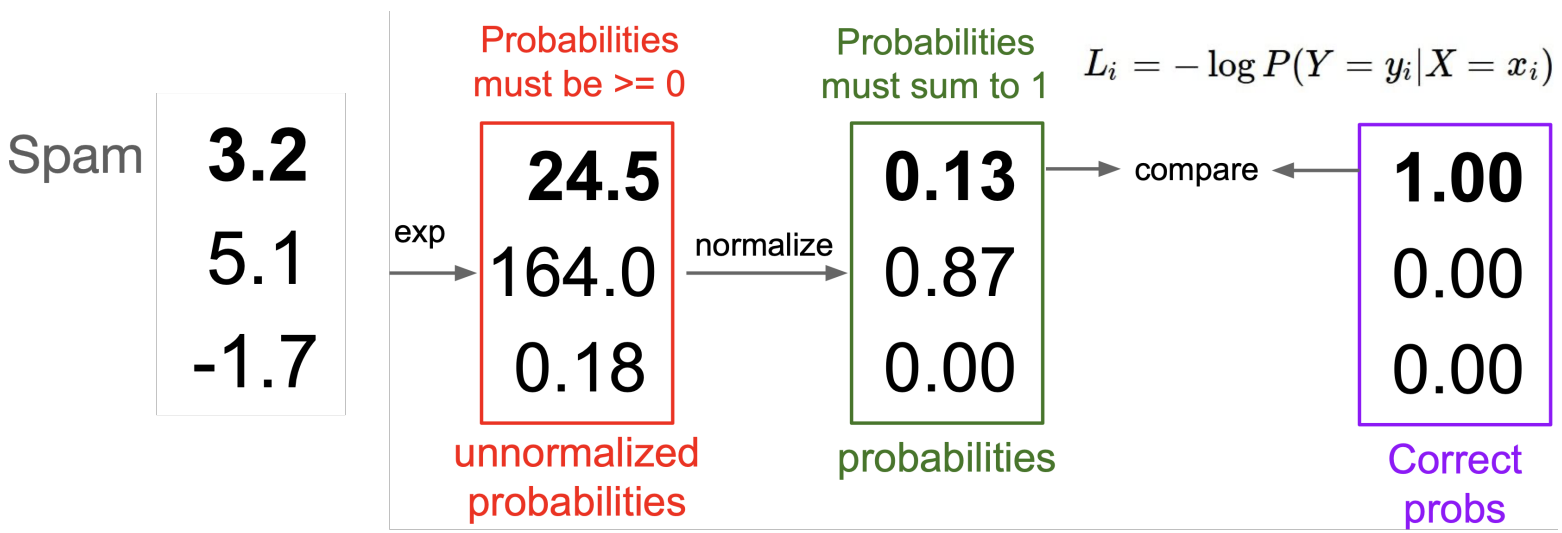


Softmax Classifier (Logistic Regression)

$$s = f(x_i; W) = Wx_i$$

Want to interpret raw classifier scores as probabilities

$$P(Y = k \mid X = x_i) = \underbrace{\frac{e^{s_k}}{\sum_j e^{s_j}}}_{\text{Softmax function}}$$



Maximizing the Likelihood of the Data

What we are doing is **maximum likelihood estimation** of the parameters of the model.

Maximum-likelihood estimates given training sample (x_i, y_i) for $i = 1, \ldots, n$:

$$\mathbf{W}_{ML} = \underset{\mathbf{W} \in \mathbb{R}^{\mathcal{Y} \times C}}{\arg\max}\, \mathcal{L}(\mathbf{W})$$

$L(\mathbf{W}) = \sum_{i=1}^{n} \log p\left(y_i \mid x_i; \mathbf{W}\right)$

$$p(y \mid x; \mathbf{W}) = \frac{\exp(\mathbf{W}_{\mathbf{y}} \cdot x)}{\sum_{y' \in \mathcal{Y}} \exp\left(\mathbf{W}_{\mathbf{y}'} \cdot x\right)}$$

Maximizing the Likelihood of the Data

Maximum-likelihood estimates given training sample (x_i, y_i) for $i = 1, \ldots, n$:

$$\mathbf{W}_{ML} = \arg\max_{\mathbf{W} \in \mathbb{R}^{\mathcal{Y} \times C}} \mathcal{L}(\mathbf{W})$$

$$p(y \mid x; \mathbf{W}) = \frac{\exp(\mathbf{W}_{\mathbf{y}} \cdot x)}{\sum_{y' \in \mathcal{Y}} \exp\left(\mathbf{W}_{\mathbf{y}'} \cdot x\right)}$$

$$L(\mathbf{W}) = \sum_{i=1}^{n} \log p\left(y_i \mid x_i; \mathbf{W}\right)$$

$$= \sum_{i=1}^{n} \mathbf{W}_{y_i} \cdot x - \sum_{i=1}^{n} \log \sum_{y' \in \mathcal{Y}} \exp\left(\mathbf{W}_{y'} \cdot x\right)$$

Basic (supervised) Machine Learning Recipe

Given: (many) input / output examples (x, y)

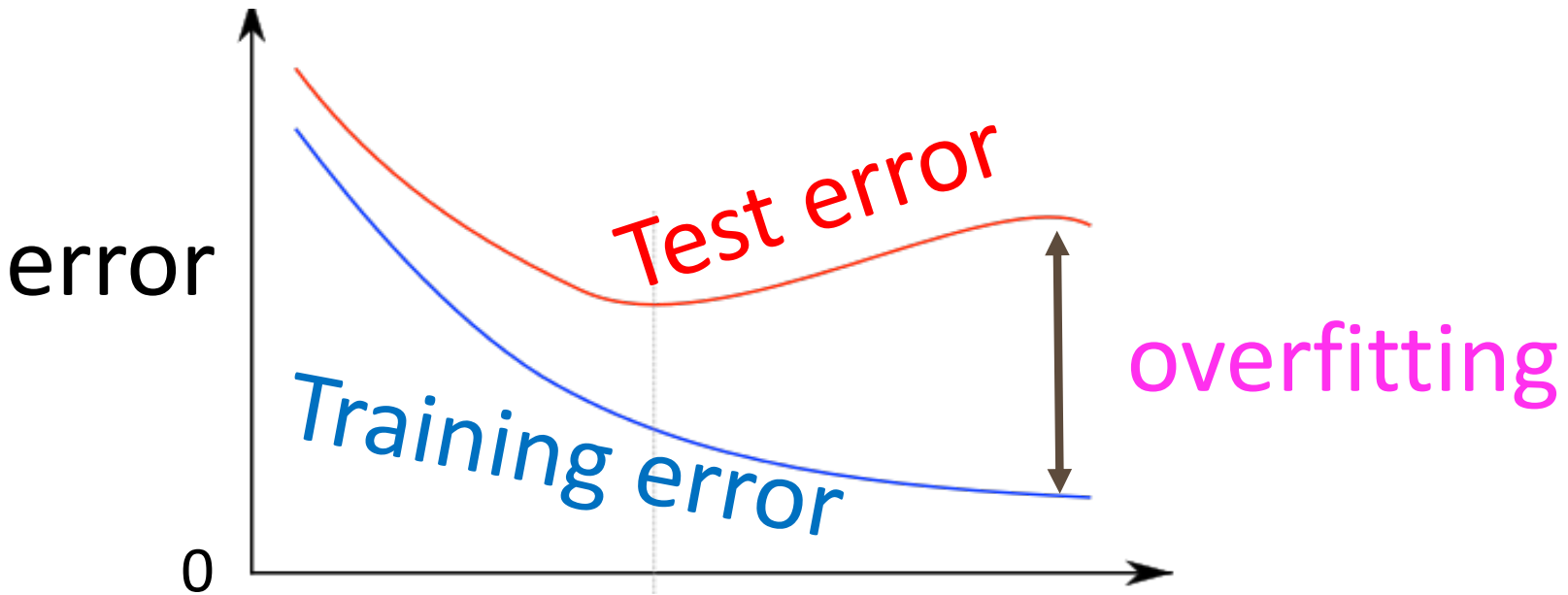
Find mapping f s.t. $\mathbf{y} \approx f(\mathbf{x})$

Define $L(\boldsymbol{W}) = \sum_{i=1}^{N} L\left(f_{\boldsymbol{w}}, \{\mathbf{x}_i, \mathbf{y}_i\}\right)$

and learn by optimizing $\boldsymbol{W} = \arg\min_{\boldsymbol{W}} L(\boldsymbol{W})$

Remember: we want model to generalize

- **Generalization**: ability to perform well on unseen data
- **Regularization**: prevents overfitting by discourageing the model from fitting the training data too closely
  - especially in the face of lots of features or very deep models



Regularization

Regularization is very important; we don't actually do Maximum Likelihood Estimation (MLE)!

Maximize probability of correct class

$$L_i = -\log P\left(Y = y_i \mid X = x_i\right)$$

Putting it all together:

$$L_i = -\log\left(\frac{e^{s^{y_i}}}{\sum_j e^{s^{y_j}}}\right)$$

Regularization

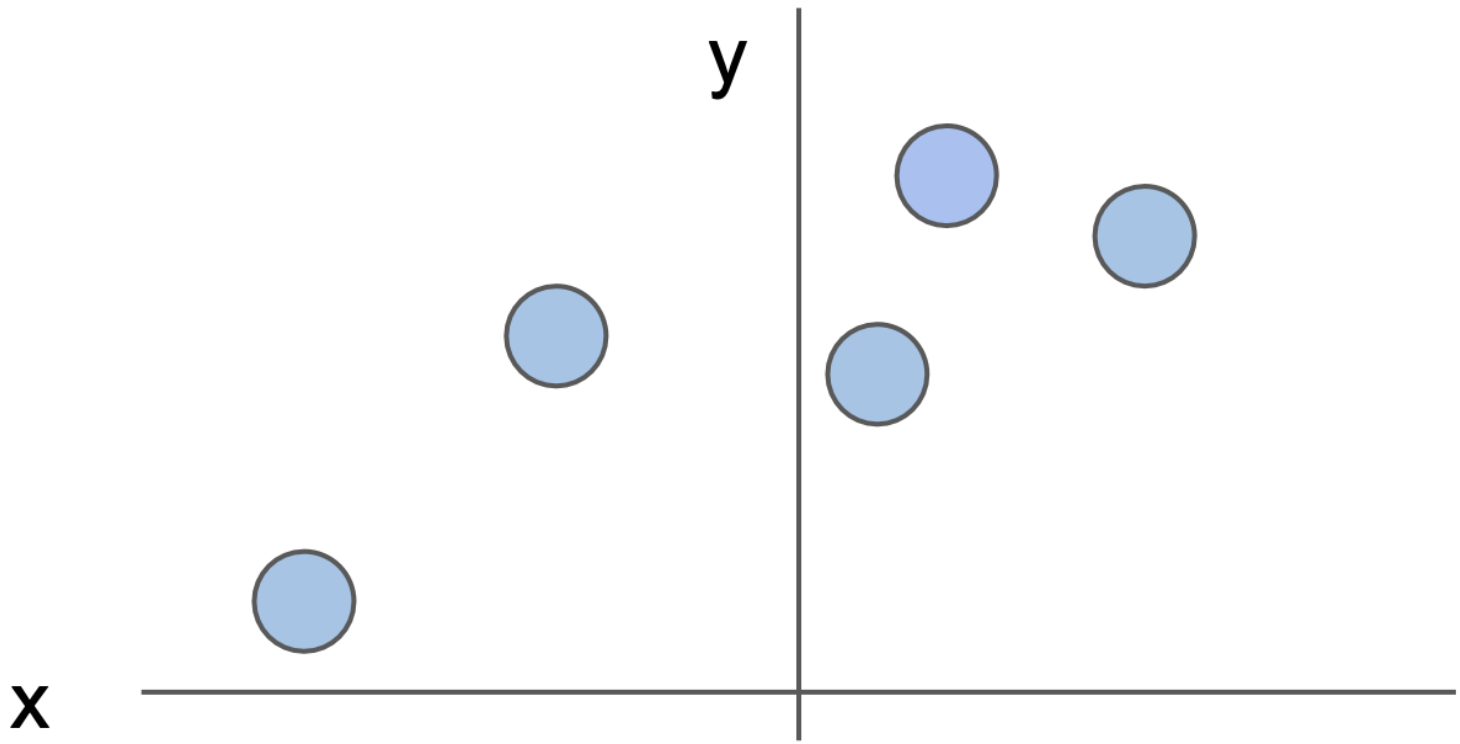
$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i\left(f(x_i; W), y_i\right) + \lambda R(W)$$

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

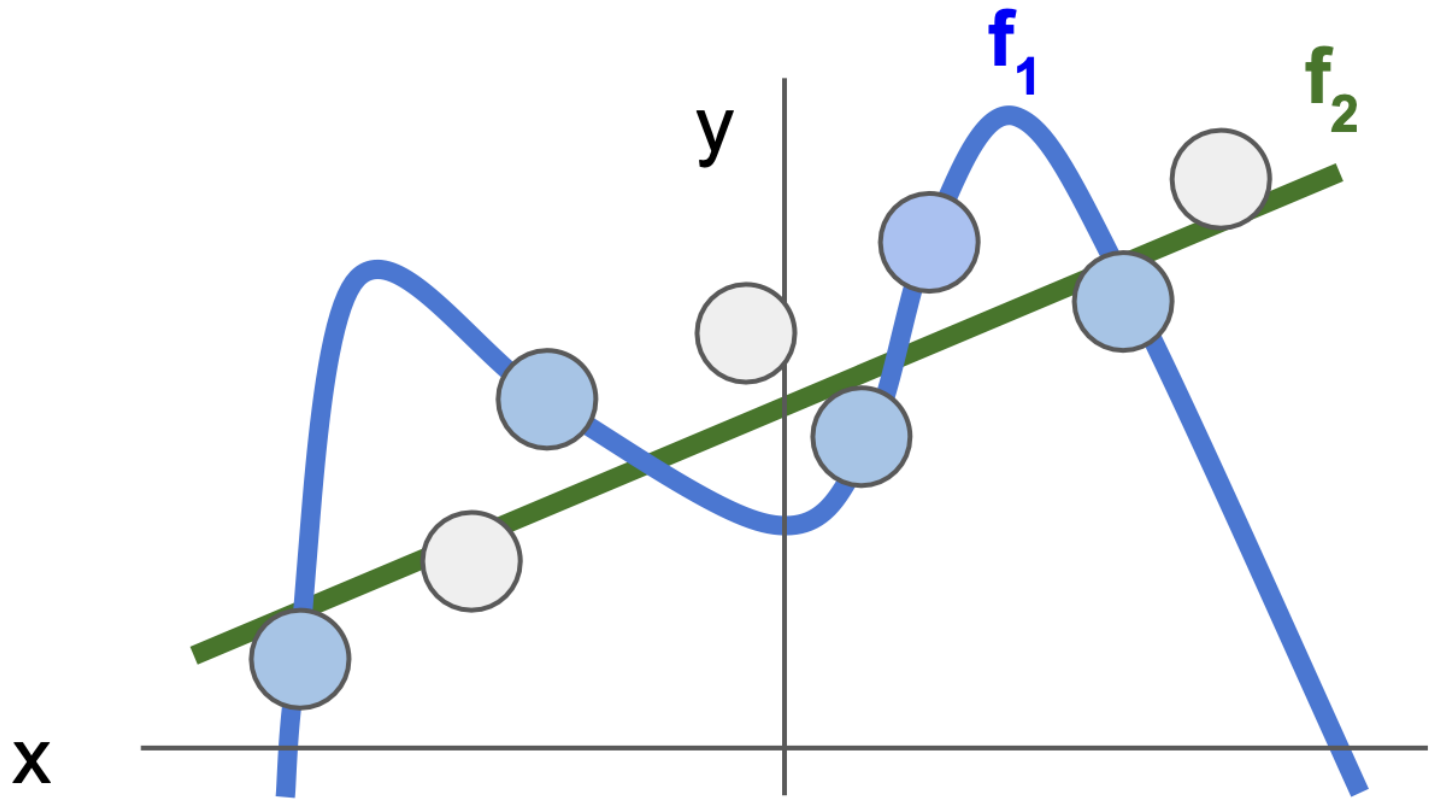
Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Regularization intuition: toy example training data



Regularization intuition: Prefer Simpler Models



- Regularization pushes against fitting the data too well so we don't fit noise in the data

Regularization Strength

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

λ = regularization strength (hyperparameter)

- $\lambda = 0$: No regularization
- $\lambda = \infty$: Only care about regularization
- $\lambda = 0.5$: Balance between fitting the data and regularization

Regularization: Methods

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

λ = regularization strength (hyperparameter)

Simple Regularization methods:

- L2: $R(W) = \sum_k \sum_l W_{k,l}^2$
- L1: $R(W) = \sum_k \sum_l |W_{k,l}|$

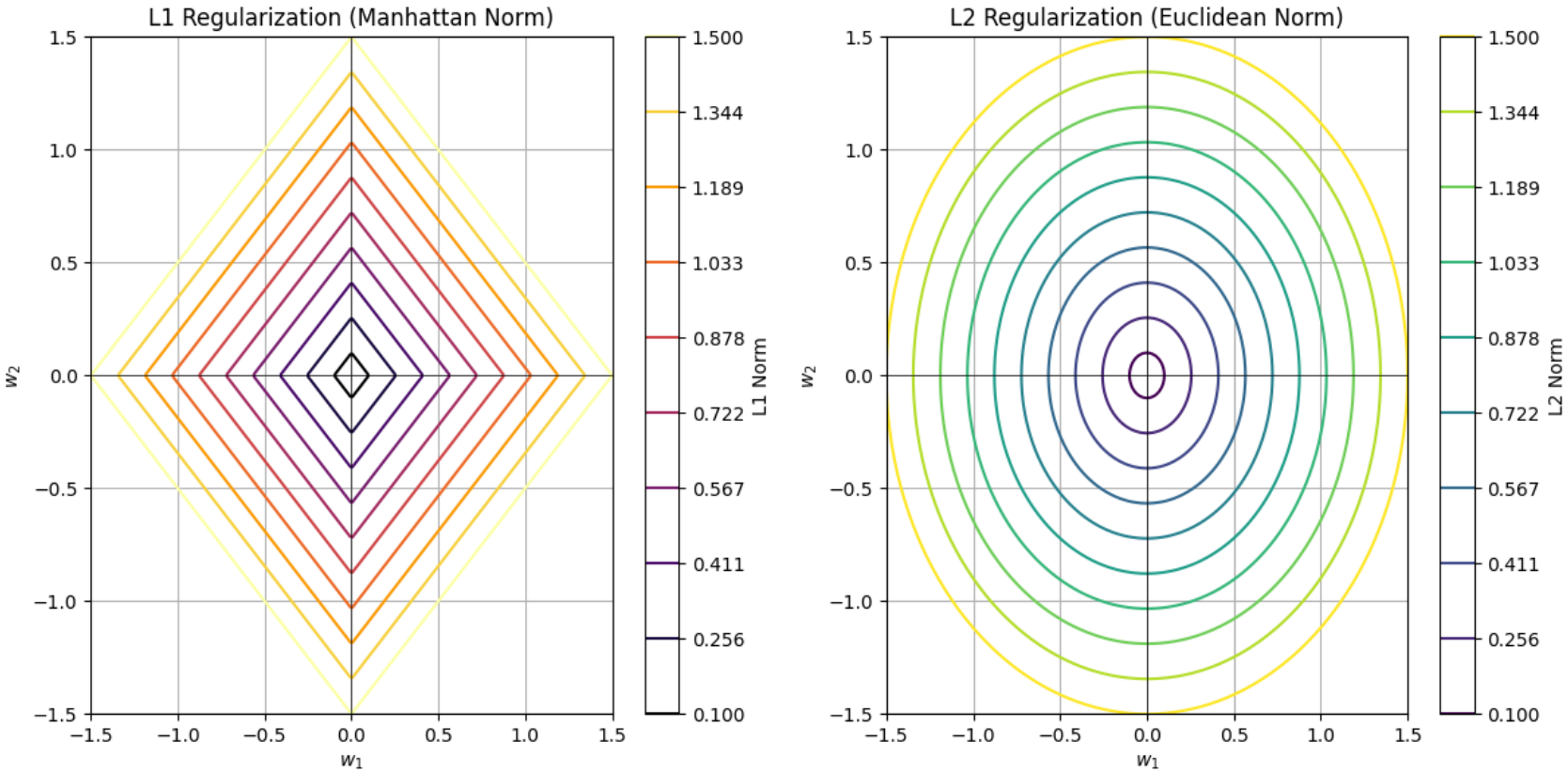
Complex methods:

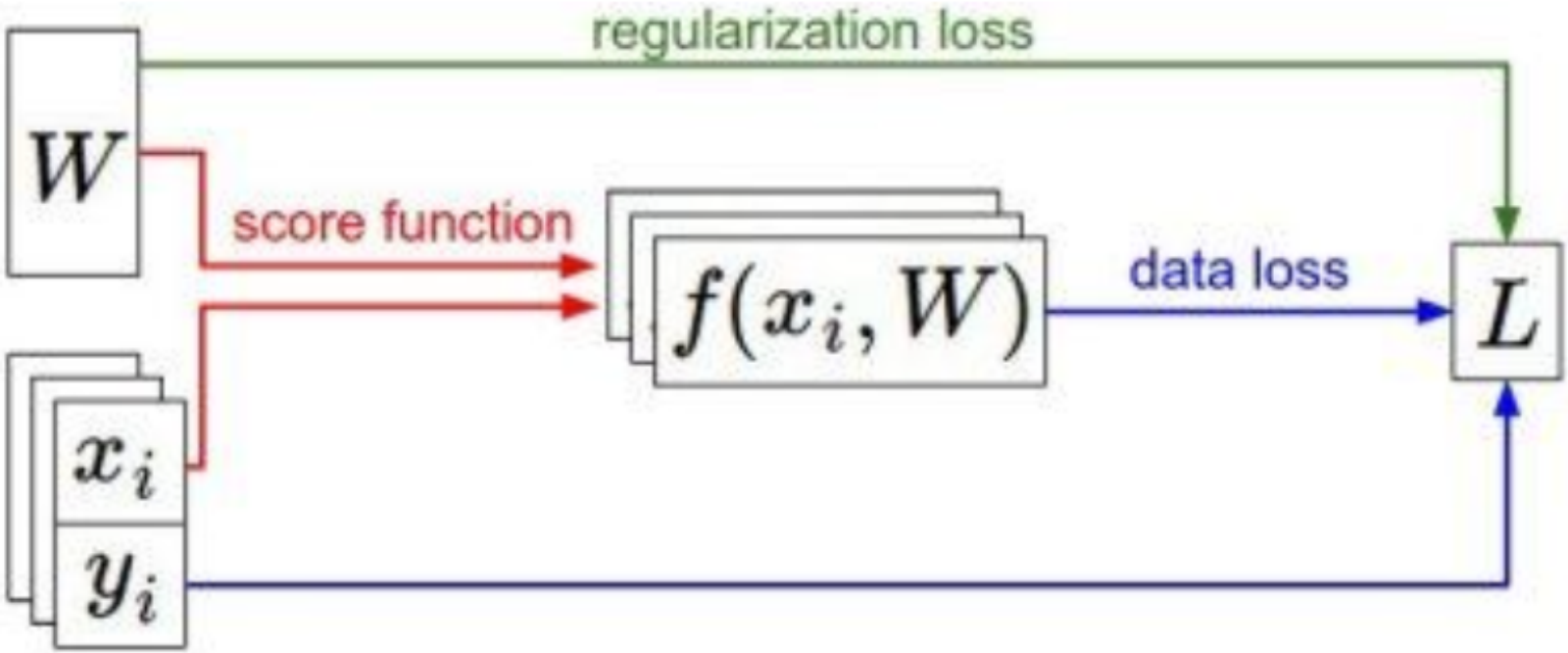
- Dropout
- Batch normalization

Regularization: Why regularize

- Express preferences over weights
- Make the model simple so it works on test data (Prevent fitting noise in the data)
- Improve optimization by adding curvature

Regularization: Expressing preferences




regularization loss
W
score function
$f(x_i, W)$
data loss
L
x_i
y_i

Basic (supervised) Machine Learning Recipe

Given: (many) input / output examples (x, y)

Find mapping f s.t. $\mathbf{y} \approx f(\mathbf{x})$

Define $L(\boldsymbol{W}) = \sum_{i=1}^{N} L\left(f_{\boldsymbol{w}}, \{\mathbf{x}_i, \mathbf{y}_i\}\right) + R(W)$

learn by **o**ptimizing $\boldsymbol{W} = \arg\min_{\boldsymbol{W}} L(\boldsymbol{W})$

How does one pick the best $\mathbf{W}$?

Questions ?

Many slides are based on slides from the Stanford CS231n course by Fei-Fei Li et al.