# Convolutional Neural Networks & Classification

# Data Science in Electron Microscopy

**Philipp Pelz**

**2024**

https://github.com/ECLIPSE-Lab/SS24_DataScienceForEM

# Regression

- Used to answer *how much?* or *how many?* questions.
  - Predicting house prices.
  - Estimating baseball team wins.
  - Forecasting patient hospital stay duration.
- Important distinctions within regression:
  - House price: Non-negative, often relative to baseline price.
    - Regress on logarithm of price.
  - Hospital stay duration: Discrete, nonnegative.
    - Least mean squares may not be ideal.
    - Specialized subfield: *survival modeling*.

# Key Points

- Estimation involves more than minimizing squared errors.
- Supervised learning encompasses more than regression.

## Focus of this Section

- Classification problems.
  - Shift focus from *how much?* to *which category?* questions.

## Classification Examples

- Does this email belong in the spam folder or the inbox?
- Is this customer more likely to sign up or not to sign up for a subscription service?
- Does this image depict a donkey, a dog, a cat, or a rooster?
- Which movie is Aston most likely to watch next?
- Which section of the book are you going to read next?

# Understanding Classification

- Colloquially, *classification* refers to:
  1. Hard assignments of examples to categories (classes).
  2. Soft assignments, assessing the probability of each category.
- Distinction between hard and soft assignments often blurred:
  - Models making soft assignments used even for hard assignment tasks.

## Multi-Label Classification

- Some cases involve more than one true label:
  - Example: A news article covering entertainment, business, and space flight, but not medicine or sports.
- Known as multi-label classification(https://en.wikipedia.org/wiki/

# Simple Image Classification Problem

## Problem Setup

- **Input**: grayscale image.
  - Each pixel value represented by a scalar.
  - Four features: $x_1, x_2, x_3, x_4$.
- **Categories**: “cat”, “chicken”, “dog”.

## Label Representation

## Natural Impulse

- Use  for  respectively.
  - $y \in \{\text{dog}, \text{cat}, \text{chicken}\}$
  - Efficient for storage.
  - Suitable for ordinal data with natural ordering, e.g., age categories.

## Ordinal Regression

- Useful for naturally ordered categories.
- Resources:
  - Overview of ranking loss functions: :citet:`Moon.Smola.Chang.ea.2010`.
  - Bayesian approach for multi-modal responses: :citet:`Beutel.Murray.Faloutsos.ea.2014`.

# One-Hot Encoding

- Suitable for unordered categorical data.
- Vector with components for each category.
- Example:
  - “cat” $\rightarrow$ $(1, 0, 0)$
  - “chicken” $\rightarrow$ $(0, 1, 0)$
  - “dog” $\rightarrow$ $(0, 0, 1)$

$$y \in \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}.$$

# Linear Model

## Estimating Conditional Probabilities

- **Objective**: Estimate conditional probabilities for all classes.
- **Model Requirement**: Multiple outputs (one per class).

## Affine Functions for Classification

- Need as many affine functions as output categories.
- For symmetry, use redundant parametrization:
  - Last category derived from others.
- **Case Example**:
  - 4 features, 3 output categories.
  - 12 scalars for weights () and 3 scalars for biases ().

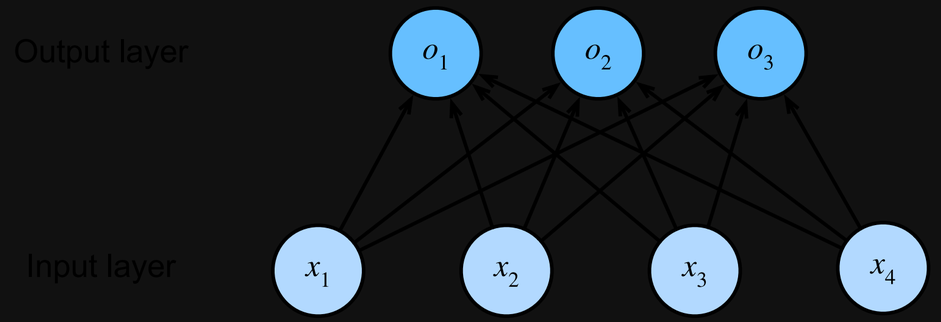
$w_{ij}$ $b_i$

# Model Equations

$$
\begin{aligned}
o_1 &= x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1, \\
o_2 &= x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2, \\
o_3 &= x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3.
\end{aligned}
$$

# Neural Network Diagram

- **Diagram**: See :numref:`fig_softmaxreg`.
- **Structure**:
  - Single-layer neural network (similar to linear regression).
  - Each output () depends on all inputs ($x_1, x_2, , ).
  - Output layer described as a *fully connected layer*.$o_1, o_2, o_3$$x_4$




fig_softmaxreg

# Concise Notation with Vectors and Matrices

- Use vectors and matrices for concise notation:
  - $\mathbf{o} = \mathbf{W}\mathbf{x} + \mathbf{b}$
  - Suited for mathematics and code implementation.
- **Weights and Biases**:
  - Weights gathered into a $3 \times 4$ matrix.
  - Biases $\mathbf{b} \in \mathbb{R}^3$ in a vector.

# The Softmax

:label:`subsec_softmax_operation`

## Limitations of Vector-Valued Regression

- Directly minimizing the difference between $\mathbf{o}$ and labels $\mathbf{y}$ has drawbacks:
  - No guarantee that outputs $o_i$ sum to 1 (expected for probabilities).
  - Outputs $o_i$ might be negative or exceed 1 even if they sum to 1.
- These issues complicate the estimation problem and make solutions brittle to outliers.

## Example Problem

- Positive linear dependency between number of bedrooms and likelihood of buying a house:
  - Probability might exceed 1 for a mansion.

# Solution: Squish the Outputs

## Probit Model

- Assume outputs $\mathbf{o}$ are corrupted versions of $\mathbf{y}$:
  - $\mathbf{y} = \mathbf{o} + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$.
  - Known as the probit model, introduced by :citet:`Fechner.1860`.
  - Less effective and harder to optimize compared to softmax.

## Softmax Function

- Use an exponential function: $P(y = i) \propto \exp o_i$.
  - Ensures nonnegative probabilities.
  - Monotonic relationship: Conditional class probability increases with $o_i$.
- Normalize to ensure probabilities sum to 1:
  - Divide each exponential value by their sum.

# Softmax Function

:eqlabel:`eq_softmax_y_and_o`

$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}) \quad \text{where} \quad \hat{y}_i = \frac{\exp(o_i)}{\sum_j \exp(o_j)}.$$

- The largest coordinate of $\mathbf{o}$ corresponds to the most likely class according to $\hat{\mathbf{y}}$.
- Softmax preserves the ordering among its arguments:
  - To determine the highest probability class, compute:

$$\operatorname*{argmax}_j \hat{y}_j = \operatorname*{argmax}_j o_j.$$

# Vectorization

## Computational Efficiency with Vectorization

- **Minibatch**:
  - examples, each with inputs. $\mathbf{X} \in \mathbb{R}^{n \times d}$
- **Output Categories**: categories
  - Weights: $\mathbf{W} \in \mathbb{R}^{d \times q}$
  - Bias: $\mathbf{b} \in \mathbb{R}^{1 \times q}$

$n$ $d$ $q$

# Vectorized Equations

$$\mathbf{O} = \mathbf{X}\mathbf{W} + \mathbf{b},$$
$$\hat{\mathbf{Y}} = \mathrm{softmax}(\mathbf{O}).$$

`eq_minibatch_softmax_reg`

- **Efficiency**: Converts the dominant operation to matrix-matrix product $\mathbf{X}\mathbf{W}$.
- **Rowwise Softmax**:
  - Exponentiate entries in each row of $\mathbf{O}$.
  - Normalize by the sum for each row.

# Numerical Stability

- **Care Needed**: Avoid overflow/underflow with large numbers.
- **Deep Learning Frameworks**: Handle these issues automatically.



# Loss Function

## Mapping Features to Probabilities

- Goal: Optimize the accuracy of mapping from features $\mathbf{x}$ to probabilities $\hat{\mathbf{y}}$.
- Method: Maximum likelihood estimation.
  - Same concept as used for probabilistic justification of mean squared error loss in :numref:`subsec_normal_distribution_and_squared_loss`.

# Log-Likelihood

- **Softmax Output**: Vector $\hat{\mathbf{y}}$ interpreted as estimated conditional probabilities:
  - Example: $\hat{y}_1 = P(y = \text{cat} \mid \mathbf{x})$.
- **Dataset Representation**:
  - Features: $\mathbf{X}$
  - Labels: $\mathbf{Y}$ (one-hot encoded).
- **Probability Comparison**:
  - Compare estimated probabilities with actual classes:

$$P(\mathbf{Y} \mid \mathbf{X}) = \prod_{i=1}^{n} P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}).$$

- **Factorization**: Assumes each label is drawn independently

# Negative Log-Likelihood

$$-\log P(\mathbf{Y} \mid \mathbf{X}) = \sum_{i=1}^{n} -\log P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}) = \sum_{i=1}^{n} l(\mathbf{y}$$

- **Loss Function**: For any pair of label $\mathbf{y}$ and model prediction $\hat{\mathbf{y}}$ over $q$ classes.

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j.$$

:eq|label:eq_1_cross_entropy

# Cross-Entropy Loss

- **Definition**: Loss function in :eqref:`eq_l_cross_entropy` is called *cross-entropy loss*.
- **One-Hot Vector**: $\mathbf{y}$ is a one-hot vector of length $q$.
  - Sum over all coordinates $j$ vanishes for all but one term.

# Properties of Cross-Entropy Loss

- **Bounded Below by 0**: Loss $l(\mathbf{y}, \hat{\mathbf{y}})$ is bounded from below by $0$ when $\hat{y}$ is a probability vector.
  - No single entry larger than $1$, negative logarithm cannot be lower than $0$.
  - $l(\mathbf{y}, \hat{\mathbf{y}}) = 0$ only if the actual label is predicted with certainty.
  - Predicting with certainty requires input $o_i$ to go to infinity (or others to negative infinity).
- **Infinite Loss**: Assigning a probability of $0$ incurs infinite loss ($-\log 0 = \infty$).
  - High confidence errors result in infinite loss.

# Softmax and Cross-Entropy Loss

:label:subsec_softmax_and_derivatives

## Calculation of Softmax and Cross-Entropy Loss

- **Combining Equations**:
  - Plugging :eqref:eq_softmax_y_and_o into the definition of the loss in :eqref:eq_l_cross_entropy:

$$
\begin{aligned}
l(\mathbf{y}, \hat{\mathbf{y}}) &= -\sum_{j=1}^{q} y_j \log \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} \\
&= \sum_{j=1}^{q} y_j \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j \\
&= \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j.
\end{aligned}
$$

# Derivative with Respect to Logit

$$\partial_{o_j} l(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} - y_j = \mathrm{softmax}(\mathbf{o})_j - y_j$$

- **Interpretation**:
  - Derivative is the difference between model probability (softmax) and actual label (one-hot vector).
  - Similar to regression where gradient was the difference between observation $y$ and estimate $\hat{y}$.
  - In exponential family models, gradients of log-likelihood are given by this term, making gradient computation easy.

# Distribution Over Outcomes

- **General Case**:
  - Observe entire distribution over outcomes.
  - Label $\mathbf{y}$ as a probability vector (e.g., $(0.1, 0.2, 0.7)$) instead of binary vector.
- **Mathematical Consistency**:
  - The same loss function $l$ in :eqref:`eq_l_cross_entropy` applies.
  - Represents expected value of the loss for a distribution over labels.

# Cross-Entropy Loss

# Information Theory Basics

:label:`subsec_info_theory_basics`

Many deep learning papers use intuition and terms from information theory. To make sense of them, we need some common language. This is a survival guide. *Information theory* deals with the problem of encoding, decoding, transmitting, and manipulating information (also known as data).

## Entropy

- **Central Idea**: Quantify the amount of information contained in data, placing a limit on data compression.
- **Definition**: For a distribution , its *entropy* is:

$$P$$

$$H[P] = \sum_j -P(j) \log P(j).$$

`eq_softmax_reg_entropy`

- **Fundamental Theorem**: To encode data drawn randomly from distribution , we need at least "nats" to encode it :cite:`Shannon.1948`.
  - **Nat**: Equivalent of a bit but using base instead of base 2.
  - 1 nat ≈ 1.44 bits.

## Surprisal

- **Compression and Prediction**: Easy-to-predict data is easy to compress.
  - Example: A data stream where every token is the same is easy to predict and compress.
- **Surprise**: Greater when an event has a lower assigned probability.
  - Claude Shannon quantified *surprisal* at observing an event with probability as .
- **Expected Surprisal**: Entropy defined in :eqref:`eq_softmax_reg_entropy` is the *expected surprisal* when probabilities match the data-generating process.

# Cross-Entropy Revisited

## Definition and Interpretation

- **Entropy**: The level of surprise experienced by someone who knows the true probability.
- **Cross-Entropy**: The expected surprisal of an observer with subjective probabilities $Q$ upon seeing data generated according to probabilities $P$.
  - Denoted as $H(P, Q)$.
  - Formula: $H(P, Q) \stackrel{\text{def}}{=} \sum_j -P(j) \log Q(j)$.
  - Lowest cross-entropy achieved when $Q = P$: $H(P, Q) = H(P)$.

## Cross-Entropy in Classification

- Two perspectives on cross-entropy classification objective:
  1. Maximizing the likelihood of observed data.
  2. Minimizing surprisal (and thus the number of bits) required to communicate the labels.

# Summary and Discussion

## Key Points

- Introduced the first nontrivial loss function for optimizing over *discrete* output spaces.
- Used a probabilistic approach, treating discrete categories as instances of draws from a probability distribution.
- Encountered the softmax activation function, transforming neural network outputs into valid discrete probability distributions.
- Derivative of cross-entropy loss combined with softmax is similar to the derivative of squared error (difference between expected behavior and prediction).
- Briefly touched on connections to statistical physics and information theory.

# Exercises

1. We can explore the connection between exponential families and the softmax in some more depth.
   1. Compute the second derivative of the cross-entropy loss $l(\mathbf{y}, \hat{\mathbf{y}})$ for the softmax.
   2. Compute the variance of the distribution given by $\mathrm{softmax}(\mathbf{o})$ and show that it matches the second derivative computed above.
2. Assume that we have three classes which occur with equal probability, i.e., the probability vector is $(\frac{1}{3}, \frac{1}{3},$.
   1. What is the problem if we try to design a binary code for it?
   2. Can you design a better code? Hint: what happens if we try to encode two independent observations? What if we encode $n$ observations jointly?
3. When encoding signals transmitted over a physical wire, engineers do not always use binary codes. For instance, PAM-3 uses three signal levels $\{-1, 0, 1\}$ as opposed to two levels $\{0, 1\}$. How many ternary units do you need to transmit an integer in the range $\{0, \ldots, 7\}$? Why might this be a better idea in terms of electronics?

1. The Bradley-Terry model uses a logistic model to capture preferences. For a user to choose between apples and oranges one assumes scores $o_{\text{apple}}$ and $o_{\text{orange}}$. Our requirements are that larger scores should lead to a higher likelihood in choosing the associated item and that the item with the largest score is the most likely one to be chosen :cite:`Bradley.Terry.1952`.
   1. Prove that the softmax satisfies this requirement.
   2. What happens if you want to allow for a default option of choosing neither apples nor oranges? Hint: now the user has 3 choices.
2. Softmax derives its name from the following mapping: $\mathrm{RealSoftMax}(a, b) = \log(\exp(a) + \exp(b))$.
   1. Prove that $\mathrm{RealSoftMax}(a, b) > \max(a, b)$.
   2. How small can you make the difference between both functions? Hint: without loss of generality you can set $b = 0$ and $a \geq b$.
   3. Prove that this holds for $\lambda^{-1}\mathrm{RealSoftMax}(\lambda a, \lambda b)$, provided that $\lambda > 0$.
   4. Show that for $\lambda \to \infty$ we have $\lambda^{-1}\mathrm{RealSoftMax}(\lambda a, \lambda b) \to \max(a, b)$.
   5. What does the soft-min look like?
   6. Extend this to more than two numbers.

1. The function $g(\mathbf{x}) \stackrel{\text{def}}{=} \log \sum_i \exp x_i$ is sometimes also referred to as the log-partition function.
   1. Prove that the function is convex. Hint: to do so, use the fact that the first derivative amounts to the probabilities from the softmax function and show that the second derivative is the variance.
   2. Show that $g$ is translation invariant, i.e., $g(\mathbf{x} + b) = g(\mathbf{x})$.
   3. What happens if some of the coordinates $x_i$ are very large? What happens if they're all very small?
   4. Show that if we choose $b = \max_i x_i$ we end up with a numerically stable implementation.
2. Assume that we have some probability distribution $P$. Suppose we pick another distribution $Q$ with $Q(i) \propto P(i)^\alpha$ for $\alpha$.
   1. Which choice of $\alpha$ corresponds to doubling the temperature? Which choice corresponds to halving it?
   2. What happens if we let the temperature converge to $0$?
   3. What happens if we let the temperature converge to $\infty$?

# The Image Classification Dataset

## MNIST Dataset

- **Overview**: Widely used for image classification of handwritten digits.
  - **Reference**: MNIST dataset :cite:`LeCun.Bottou.Bengio.ea.1998`.
  - **Content**: 60,000 images of $28 \times 28$ pixels resolution (plus a test dataset of 10,000 images).
- **Historical Context**:
  - Released in the 1990s, it was a formidable challenge for machine learning algorithms.
  - Example of state-of-the-art equipment at the time: Sun SPARCStation 5 with 64MB RAM and 5 MFLOPs (AT&T Bell Laboratories, 1995).
- **Impact**:
  - High accuracy on digit recognition was crucial for automating letter sorting for the USPS in the 1990s.
  - Algorithms achieving error rates below 1%:
    - Deep networks like LeNet-5 :cite:`LeCun.Jackel.Bottou.ea.1995`.
    - Support vector machines with invariances :cite:`Scholkopf.Burges.Vapnik.1996`.
    - Tangent distance classifiers :cite:`Simard.LeCun.Denker.ea.1998`.

# MNIST as a Benchmark

- **Historical Significance**: Served as the reference for comparing machine learning algorithms for over a decade.
- **Current Relevance**:
  - Simple models achieve classification accuracy over 95%.
  - Unsuitable for distinguishing between stronger and weaker models due to high levels of accuracy.
  - Skewed algorithmic development towards methods that excel with clean datasets.

# Transition to Fashion-MNIST

- **Limitations of MNIST**: More of a sanity check than a benchmark today.
- **ImageNet**:
  - Poses a more relevant challenge :cite:`Deng.Dong.Socher.ea.2009`.
  - Too large for interactive examples and illustrations in this context.
- **Fashion-MNIST**:
  - Released in 2017 :cite:`Xiao.Rasul.Vollgraf.2017`.
  - Contains images of 10 categories of clothing at $28 \times 28$ pixels resolution.
  - Used as a substitute for MNIST in upcoming sections.

```
%matplotlib inline
import time
from d2l import torch as d2l
import torch
import torchvision
from torchvision import transforms

d2l.use_svg_display()
```

# Loading the Dataset

Since it is such a frequently used dataset, all major frameworks provide preprocessed versions of it. We can [**download and read the Fashion-MNIST dataset into memory using built-in framework utilities.**]

```
class FashionMNIST(d2l.DataModule):  #@save
    """The Fashion-MNIST dataset."""
    def __init__(self, batch_size=64, resize=(28, 28)):
        super().__init__()
```

```
        self.save_hyperparameters()
        trans = transforms.Compose([transforms.Resize(resize),
                                    transforms.ToTensor()])
        self.train = torchvision.datasets.FashionMNIST(
            root=self.root, train=True, transform=trans, download=True)
        self.val = torchvision.datasets.FashionMNIST(
            root=self.root, train=False, transform=trans, download=True)
```

# Fashion-MNIST Dataset

- **Content**: Images from 10 categories.
  - Each category represented by:
    - 6,000 images in the training dataset.
    - 1,000 images in the test dataset.
- **Dataset Sizes**:
  - **Training Set**: 60,000 images.
  - **Test Set**: 10,000 images.
- **Purpose of Test Dataset**: Used exclusively for evaluating model performance, not for training.

```
data = FashionMNIST(resize=(32, 32))
len(data.train), len(data.val)
```

# Image Characteristics

- **Grayscale Images**: Upscaled to $32 \times 32$ pixels resolution.
  - Similar to original MNIST dataset (binary black and white images).
- **Modern Image Data**:
  - Typically has 3 channels (red, green, blue).
  - Hyperspectral images can have over 100 channels (e.g., HyMap sensor has 126 channels).
- **Storage Convention**:
  - Images stored as a $c \times h \times w$ tensor.
  - $c$ = number of color channels.
  - $h$ = height.
  - $w$ = width.

```
data.train[0][0].shape
```

The categories of Fashion-MNIST have human-understandable names. The following convenience method converts between numeric labels and their names.

```
@d2l.add_to_class(FashionMNIST)  #@save
def text_labels(self, indices):
    """Return text labels."""
    labels = ['t-shirt', 'trouser', 'pullover', 'dress', 'coat',
              'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
    return [labels[int(i)] for i in indices]
```

# Reading a Minibatch

- **Data Iterator**: Utilizes built-in data iterator for reading from training and test sets.
- **Minibatch Size**: At each iteration, a data iterator reads a minibatch of data with size `batch_size`.
- **Shuffling**: Randomly shuffle the examples for the training data iterator to ensure better training performance.

```
@d2l.add_to_class(FashionMNIST)  #@save
def get_dataloader(self, train):
    data = self.train if train else self.val
    return torch.utils.data.DataLoader(data, self.batch_size, shuffle=train,
                                       num_workers=self.num_workers)
```

To see how this works, let's load a minibatch of images by invoking the `train_dataloader` method. It contains 64 images.

```
X, y = next(iter(data.train_dataloader()))
print(X.shape, X.dtype, y.shape, y.dtype)
```

# Performance of Data Loading

- **Time to Read Images**: Not extremely fast, but sufficient for our needs.
- **Context**: Processing images with a deep network takes significantly longer than loading them.
- **Conclusion**: The data loader's speed ensures that training a network will not be IO constrained.

```
tic = time.time()
for X, y in data.train_dataloader():
    continue
f'{time.time() - tic:.2f} sec'
```

# Visualization

- **Frequent Use**: We'll use the Fashion-MNIST dataset frequently.
- **Convenience Function**: `show_images` function to visualize images and labels.
  - Implementation details in the appendix.

```python
def show_images(imgs, num_rows, num_cols, titles=None, scale=1.5):  #@save
    """Plot a list of images."""
    raise NotImplementedError
```

```python
@d2l.add_to_class(FashionMNIST)  #@save
def visualize(self, batch, nrows=1, ncols=8, labels=[]):
    X, y = batch
    if not labels:
        labels = self.text_labels(y)
    if tab.selected('mxnet', 'pytorch'):
        d2l.show_images(X.squeeze(1), nrows, ncols, titles=labels)
    if tab.selected('tensorflow'):
        d2l.show_images(tf.squeeze(X), nrows, ncols, titles=labels)
    if tab.selected('jax'):
        d2l.show_images(jnp.squeeze(X), nrows, ncols, titles=labels)

batch = next(iter(data.val_dataloader()))
data.visualize(batch)
```

We are now ready to work with the Fashion-MNIST dataset in the sections that follow.

# Summary

# The Base Classification Model

:label:`sec_classification`

- **Observation**: Implementations from scratch and concise implementations using frameworks are similar for both regression and classification.
- **Purpose**: Since many models in this book deal with classification, we aim to add functionalities to support this setting specifically.
- **Base Class**: This section provides a base class for classification models to simplify future code.

```
from d2l import torch as d2l
import torch
```

# The `Classifier` Class

- **Definition**: `Classifier` class.
- **Validation Step**:
  - Reports both the loss value and classification accuracy on a validation batch.
  - Update every `num_val_batches` batches.
  - Generates averaged loss and accuracy on the whole validation data.
  - Average numbers may not be exact if the last batch contains fewer examples, but this is ignored for simplicity.

```python
class Classifier(d2l.Module):  #@save
    """The base class of classification models."""
    def validation_step(self, batch):
        Y_hat = self(*batch[:-1])
        self.plot('loss', self.loss(Y_hat, batch[-1]), train=False)
        self.plot('acc', self.accuracy(Y_hat, batch[-1]), train=False)
```

By default we use a stochastic gradient descent optimizer, operating on minibatches, just as we did in the context of linear regression.

```
@d2l.add_to_class(d2l.Module)  #@save
def configure_optimizers(self):
    return torch.optim.SGD(self.parameters(), lr=self.lr)
```

# Accuracy

- **Predicted Probability Distribution**: `y_hat`
  - Choose the class with the highest predicted probability for hard predictions.
  - Example: Gmail categorizing emails into “Primary”, “Social”, “Updates”, “Forums”, or “Spam”.
- **Correct Predictions**: When predictions match the label class `y`.
  - **Classification Accuracy**: Fraction of correct predictions.
  - Difficult to optimize directly as it is not differentiable, but crucial for performance measurement and benchmarks.
  - Almost always reported when training classifiers.



- **Computing Accuracy**:
  - Assume `y_hat` is a matrix with prediction scores for each class in the second dimension.
  - Use `argmax` to get the predicted class index for the largest entry in each row.
  - Compare predicted class with ground-truth `y` elementwise.
  - Convert `y_hat`'s data type to match `y` to ensure correct comparisons.
  - Result: Tensor with 0 (false) and 1 (true) entries.
  - Sum the tensor to get the number of correct predictions.

```python
@d2l.add_to_class(Classifier)  #@save
def accuracy(self, Y_hat, Y, averaged=True):
    """Compute the number of correct predictions."""
    Y_hat = d2l.reshape(Y_hat, (-1, Y_hat.shape[-1]))
    preds = d2l.astype(d2l.argmax(Y_hat, axis=1), Y.dtype)
    compare = d2l.astype(preds == d2l.reshape(Y, -1), d2l.float32)
    return d2l.reduce_mean(compare) if averaged else compare
```

# Summary

- **Common Problem**: Classification is common enough to warrant its own convenience functions.
- **Central Importance**: *Accuracy* of the classifier.
  - Often the primary measure of performance.
- **Training Objectives**:
  - Train classifiers to optimize various objectives for statistical and computational reasons.
  - Regardless of the loss function minimized during training, assessing accuracy empirically is crucial.
- **Convenience Method**: Useful to have methods for assessing classifier accuracy.

# Exercises

1. Denote by $L_v$ the validation loss, and let $L_v^q$ be its quick and dirty estimate computed by the loss function averaging in this section. Lastly, denote by $l_v^b$ the loss on the last minibatch. Express $L_v$ in terms of $L_v^q$, $l_v^b$, and the sample and minibatch sizes.
2. Show that the quick and dirty estimate $L_v^q$ is unbiased. That is, show that $E[L_v] = E[L_v^q]$. Why would you still want to use $L_v$ instead?
3. Given a multiclass classification loss, denoting by $l(y, y')$ the penalty of estimating $y'$ when we see $y$ and given a probabilty $p(y \mid x)$, formulate the rule for an optimal selection of $y'$. Hint: express the expected loss, using $l$ and $p(y \mid x)$.

# Softmax Regression Implementation from Scratch

## Implementing Softmax Regression

- **Fundamental Concept**: Softmax regression is a foundational model.
- **Importance**: Essential to understand and implement it yourself.
- **Approach**:
  - Define softmax-specific aspects of the model.
  - Reuse components from the linear regression section, including the training loop.

```
from d2l import torch as d2l
import torch
```

# The Softmax

- **Key Concept**: Mapping from scalars to probabilities.
- **Sum Operator**: Recall the operation of the sum operator along specific dimensions in a tensor, as discussed in :numref:`subsec_lin-alg-reduction` and :numref:`subsec_lin-alg-non-reduction`.
- **Example**:
  - Given a matrix `X`, sum over all elements by default or over elements in the same axis.
  - The `axis` variable allows for computing row and column sums.

```
X = d2l.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
d2l.reduce_sum(X, 0, keepdims=True), d2l.reduce_sum(X, 1, keepdims=True)
```

- **Computing the Softmax**: Requires three steps:
  1. Exponentiation of each term.
  2. Sum over each row to compute the normalization constant for each example.
  3. Division of each row by its normalization constant, ensuring the result sums to 1.

$$\mathrm{softmax}(\mathbf{X})_{ij} = \frac{\exp(\mathbf{X}_{ij})}{\sum_k \exp(\mathbf{X}_{ik})}.$$

- **Partition Function**:
  - The (logarithm of the) denominator is called the (log) *partition function*.
  - Introduced in statistical physics to sum over all possible states in a thermodynamic ensemble.
- **Implementation**: The implementation is straightforward.

```python
def softmax(X):
    X_exp = d2l.exp(X)
    partition = d2l.reduce_sum(X_exp, 1, keepdims=True)
    return X_exp / partition  ## The broadcasting mechanism is applied here
```

- **Input Transformation**: For any input X, [**each element is turned into a non-negative number, and each row sums up to 1**], as required for a probability.
- **Caution**:
  - The code above is *not* robust against very large or very small arguments.
  - While sufficient to illustrate the concept, do *not* use this code verbatim for serious purposes.
- **Best Practice**:
  - Deep learning frameworks have built-in protections.
  - Use the built-in softmax functions in deep learning frameworks going forward.

```
X = d2l.rand((2, 5))
X_prob = softmax(X)
X_prob, d2l.reduce_sum(X_prob, 1)
```

# The Model

- **Softmax Regression Model**: We now have everything needed to implement the softmax regression model.
- **Instance Representation**:
  - Each instance represented by a fixed-length vector.
  - Raw data consists of $28 \times 28$ pixel images.
  - [**Flatten each image, treating them as vectors of length 784.**]
- **Future Improvements**:
  - Later chapters will introduce convolutional neural networks to exploit spatial structure more effectively.
- **Outputs**:
  - Number of outputs equals the number of classes.
  - (**Dataset has 10 classes, so the network has an output dimension of 10.**)

- **Weights and Biases**:
  - Weights: $784 \times 10$ matrix.
  - Biases: $1 \times 10$ dimensional row vector.
  - Initialize weights `W` with Gaussian noise.
  - Initialize biases as zeros.

```
1
2  class SoftmaxRegressionScratch(d2l.Classifier):
3      def __init__(self, num_inputs, num_outputs, lr, sigma=0.01):
4          super().__init__()
5          self.save_hyperparameters()
6          self.W = torch.normal(0, sigma, size=(num_inputs, num_outputs),
7                                requires_grad=True)
8          self.b = torch.zeros(num_outputs, requires_grad=True)
9
10     def parameters(self):
11         return [self.W, self.b]
```

# Model Implementation

- **Mapping Input to Output**:
  - The code below defines the network's mapping from input to output.
  - Each $28 \times 28$ pixel image in the batch is flattened into a vector using `reshape` before passing through the model.

```python
class SoftmaxRegression(nn.Module):
    def __init__(self, num_inputs, num_outputs):
        super(SoftmaxRegression, self).__init__()
        self.linear = nn.Linear(num_inputs, num_outputs)

    def forward(self, X):
        X = X.reshape((-1, 784))  # Flatten each image
        return self.linear(X)
```

```python
@d2l.add_to_class(SoftmaxRegressionScratch)
def forward(self, X):
    X = d2l.reshape(X, (-1, self.W.shape[0]))
    return softmax(d2l.matmul(X, self.W) + self.b)
```

# The Cross-Entropy Loss

- **Common Loss Function**:
  - The cross-entropy loss function is perhaps the most common in deep learning.
  - Widely used in classification problems, more so than regression problems.
- **Definition**:
  - Cross-entropy takes the negative log-likelihood of the predicted probability assigned to the true label.
  - Avoid Python for-loops for efficiency; use indexing instead.
  - One-hot encoding in $\mathbf{y}$ allows selecting matching terms in $\hat{\mathbf{y}}$.
- **Example**:
  - Create sample data `y_hat` with 2 examples of predicted probabilities over 3 classes and their corresponding labels `y`.
  - Correct labels: $0$ and $2$ (i.e., the first and third class).
  - Use `y` as indices of probabilities in `y_hat` to pick out terms efficiently.

```
y = d2l.tensor([0, 2])
y_hat = d2l.tensor([[0.1, 0.3, 0.6], [0.3, 0.2, 0.5]])
y_hat[[0, 1], y]
```

Now we can (**implement the cross-entropy loss function**) by averaging over the logarithms of the selected probabilities.

```python
def cross_entropy(y_hat, y):
    return -d2l.reduce_mean(d2l.log(y_hat[list(range(len(y_hat))), y]))

cross_entropy(y_hat, y)
```

```python
@d2l.add_to_class(SoftmaxRegressionScratch)
def loss(self, y_hat, y):
    return cross_entropy(y_hat, y)
```

# Training

- **Reusing `fit` Method**: Use the `fit` method defined in :numref:`sec_linear_scratch` to [**train the model with 10 epochs.**]
- **Adjustable Hyperparameters**:
  - **Number of epochs** (`max_epochs`)
  - **Minibatch size** (`batch_size`)
  - **Learning rate** (`lr`)
  - These hyperparameters are not learned during the primary training loop but influence model performance.
- **Hyperparameter Tuning**:
  - Choose values based on the *validation* split of the data.
  - Evaluate the final model on the *test* split.
- **Evaluation**:
  - Treat the test data of Fashion-MNIST as the validation set.
  - Report validation loss and validation accuracy on this split.

```
data = d2l.FashionMNIST(batch_size=256)
model = SoftmaxRegressionScratch(num_inputs=784, num_outputs=10, lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```

# Prediction

Now that training is complete, our model is ready to [**classify some images.**]

```
X, y = next(iter(data.val_dataloader()))
if tab.selected('pytorch', 'mxnet', 'tensorflow'):
    preds = d2l.argmax(model(X), axis=1)
if tab.selected('jax'):
    preds = d2l.argmax(model.apply({'params': trainer.state.params}, X), axis=1)
preds.shape
```

# Error Analysis

- **Focus**: More interested in images labeled *incorrectly*.
- **Visualization**:
  - Compare actual labels (first line of text output) with model predictions (second line of text output).

```
wrong = d2l.astype(preds, y.dtype) != y
X, y, preds = X[wrong], y[wrong], preds[wrong]
labels = [a+'\n'+b for a, b in zip(
    data.text_labels(y), data.text_labels(preds))]
data.visualize([X, y], labels=labels)
```

# Summary

- **Experience Gained**:
  - Solving linear regression and classification problems.
  - Reached the state of the art of 1960-1970s statistical modeling.
- **Next Steps**:
  - Learn how to leverage deep learning frameworks.
  - Implement models more efficiently using these frameworks.



# Exercises



1. In this section, we directly implemented the softmax function based on the mathematical definition of the softmax operation. As discussed in :numref:`sec_softmax` this can cause numerical instabilities.
   1. Test whether `softmax` still works correctly if an input has a value of ?
   2. Test whether `softmax` still works correctly if the largest of all inputs is smaller than ?
   3. Implement a fix by looking at the value relative to the largest entry in the argument.
2. Implement a `cross_entropy` function that follows the definition of the cross-entropy loss function .
   1. Try it out in the code example above.
   2. Why do you think it runs more slowly?
   3. Should you use it? In which cases would it make sense?
   4. What do you need to be careful of? Hint: consider the domain of the logarithm.
3. Is it always a good idea to return the most likely label? For example, would you do this for medical diagnosis? How would you try to address this?
4. Assume that we want to use softmax regression to predict the next word based on some features. What are some problems that might arise from a large vocabulary?
5. Experiment with the hyperparameters of the code above. In particular:
   1. Plot how the validation loss changes as you change the learning rate.
   2. Do the validation and training loss change as you change the minibatch size? How large or small do you need to go before you see an effect?

# Generalization and Overfitting

- **Focus**: Tackling multiclass classification with linear neural networks, softmax functions, and cross-entropy loss.
  - Interpreted model outputs as probabilistic predictions.
  - Derived the cross-entropy loss function (negative log likelihood).
  - Fitted the model to the training set.
- **Goal**: Learn *general patterns* assessed empirically on previously unseen data (the test set).
  - High accuracy on the training set is meaningless if it only reflects memorization.
- **Problem with Memorization**:
  - Perfect accuracy on the training set can be achieved by memorizing the dataset.
  - Memorization doesn’t help classify new examples.
  - Without further guidance, new examples might require random guessing.



- **Key Questions**:
  1. How many test examples are needed to precisely estimate the accuracy of classifiers on the underlying population?
  2. What happens if we keep evaluating models on the same test set repeatedly?
  3. Why should fitting our linear models to the training set fare better than naive memorization?

# Overfitting and Generalization

- **Previous Coverage**:
  - Basics of overfitting and generalization discussed in :numref:`sec_generalization_basics` in the context of linear regression.
- **Current Chapter**:
  - Delve deeper into foundational ideas of statistical learning theory.
  - Explore guarantees of generalization *a priori*.
- **Generalization Guarantees**:
  - For many models, we can guarantee generalization within an upper bound $\epsilon$ on the generalization gap.
  - Determine the required number of samples $n$ to ensure the empirical error lies within $\epsilon$ of the true error, for any data generating distribution.

- **Practical Utility**:
  - These guarantees are profound but limited in practical use for deep learning practitioners.
  - Ensuring generalization of deep neural networks *a priori* would require an absurd number of examples (potentially trillions).
  - In practice, deep neural networks generalize well with far fewer examples (thousands).
- **Practical Approach**:
  - Forgo *a priori* guarantees.
  - Employ methods based on past performance on similar problems.
  - Certify generalization *post hoc* through empirical evaluations.
- **Future Discussion**:
  - Revisit generalization in :numref:`chap_perceptrons`.
  - Provide an introduction to the scientific literature explaining why deep neural networks generalize in practice.

# The Test Set

- **Role of Test Sets**:
  - Used as the gold standard method for assessing generalization error.
- **Focus**:
  - Discuss properties of generalization error estimates using a fixed classifier $f$.
  - Assume a *fresh* dataset of examples $\mathcal{D} = (\mathbf{x}^{(i)}, y^{(i)})_{i=1}^{n}$ not used to train the classifier.
- **Empirical Error**:
  - Fraction of instances where the prediction $f(\mathbf{x}^{(i)})$ disagrees with the true label $y^{(i)}$.
  - Given by:

$$\epsilon_{\mathcal{D}}(f) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(f(\mathbf{x}^{(i)}) \neq y^{(i)}).$$

- **Population Error**:
  - Expected fraction of examples in the underlying population (distribution ) where the classifier disagrees with the true label. $P(X, Y)$
  - Given by:

$$\epsilon(f) = E_{(\mathbf{x},y)\sim P}\mathbf{1}(f(\mathbf{x}) \neq y) = \int\int \mathbf{1}(f(\mathbf{x}) \neq y)p($$

# Estimating Population Error

- **Population Error**:
  - $\epsilon(f)$ is the actual quantity of interest but cannot be directly observed.
  - Similar to estimating the average height in a large population without measuring every individual.
- **Empirical Error as Estimator**:
  - Test set $\mathcal{D}$ is statistically representative of the population.
  - $\epsilon_{\mathcal{D}}(f)$ serves as a statistical estimator of $\epsilon(f)$.

- **Mean Estimation**:
  - Estimating population error is a classic problem of mean estimation (see :numref:`sec_prob`).
  - Central limit theorem: With $n$ random samples drawn from any distribution with mean $\mu$ and standard deviation $\sigma$, the sample average $\hat{\mu}$ approximates a normal distribution centered at $\mu$ with standard deviation $\sigma/\sqrt{n}$
- **Convergence Rate**:
  - As the number of examples grows, test error $\epsilon_p(f)$ approaches true error $\epsilon(f)$ at a rate of $\mathcal{O}(1/\sqrt{n})$.
  - To estimate test error twice as precisely, collect four times as large a test set.
  - To reduce test error by a factor of one hundred, collect ten thousand times as large a test set.

- **Bernoulli Random Variable**:
  - Random variable $\mathbf{1}(f(X) \neq Y)$ can take values 0 and 1 (Bernoulli random variable).
  - Parameter indicating probability of value 1 is the true error rate $\epsilon(f)$.
  - Variance $\sigma^2$ of a Bernoulli distribution is $\epsilon(f)(1-\epsilon(f))$.
  - Variance is highest when the true error rate is close to 0.5 and lower when close to 0 or 1.
- **Asymptotic Standard Deviation**:
  - Asymptotic standard deviation of $\epsilon_{\mathcal{D}}(f)$ over $n$ test samples cannot be greater than $\sqrt{0.25/n}$.

# Estimating Population Error (Continued)

- **Finite Sample Considerations**:
  - Asymptotic rate characterizes behavior as test set size approaches infinity.
  - For  to approximate  within  (one standard deviation), collect roughly 2500 samples.
  - For a 95% confidence interval (), collect 10000 samples.
- **Popular Benchmarks**:
  - Test set sizes for many popular benchmarks in machine learning are around 10000 samples.
  - Improvements of  in error rates are significant, especially when error rates are close to .

$\epsilon_{\mathcal{D}}(f)$ $\epsilon(f)$ $\pm 0.01$

$\epsilon_{\mathcal{D}}(f) \in \epsilon(f) \pm 0.01$

$0.01$

$0$

- **Finite Sample Bounds**:
  - Asymptotic analysis provides general insights but for finite samples, use Hoeffding's inequality:

$$P(\epsilon_{\mathcal{D}}(f) - \epsilon(f) \geq t) < \exp\left(-2nt^2\right).$$

- **Example Calculation**:
  - To conclude with 95% confidence that $\epsilon_{\mathcal{D}}(f)$ is within $0.01$ of $\epsilon(f)$, roughly 15000 examples are needed (compared to 10000 from asymptotic analysis).
- **General Trend**:
  - Finite sample guarantees are more conservative but not drastically different from asymptotic estimates.
  - Asymptotic analysis remains useful for providing ballpark figures.

# Test Set Reuse

- **Empirical ML Research**:
  - Models developed and validated based on test set performance.
  - Evaluate test error $\epsilon_{\mathcal{P}}(f)$ for any fixed classifier $f$ to understand population error $\epsilon(f)$.
- **Training Initial Model**:
  - Train model $f_1$ with an appropriate number of test examples.
  - Use a validation set for preliminary analysis, hyperparameter tuning, and model selection.
  - Evaluate $f_1$ on the test set, report unbiased population error estimate with confidence interval.

- **Developing New Models**:
  - Develop a new model $f_2$ and tune it on the validation set.
  - Prepare to evaluate $f_2$ but realize the test set has already been used.
- **Issues with Test Set Reuse**:
  - Original test set $\mathcal{D}$ was designed for a single classifier $f$.
  - Evaluating multiple classifiers $f_1, \ldots, f_k$ on the same test set introduces the problem of false discovery.
  - With multiple classifiers, the probability of at least one misleading result increases.
  - Example: With 20 classifiers, ensuring no misleading score is challenging.
- **Multiple Hypothesis Testing**:
  - Evaluating multiple hypotheses on the same dataset can lead to misleading conclusions.
  - Persistent problem in scientific research despite extensive statistical literature.

# Concerns with Test Set Reuse

- **Distrusting Subsequent Evaluations**:
  - Initial analysis assumes the classifier was chosen without contact with the test set.
  - Testing multiple functions and using the test set performance of $f_1$ to choose $f_2$ introduces bias.
- **Adaptive Overfitting**:
  - Occurs when information from the test set leaks to the modeler.
  - Once leaked, the test set can no longer be a true test set.
  - Topic of interest in learning theory and statistics :cite:dwork2015preserving.

- **Practical Guidance**:
  - Create and use real test sets.
  - Consult test sets infrequently.
  - Account for multiple hypothesis testing when reporting confidence intervals.
  - Increase vigilance when stakes are high and dataset size is small.
- **Benchmark Challenges**:
  - Maintain several test sets.
  - Demote old test sets to validation sets after each round.

# Statistical Learning Theory

- **Importance of Test Sets**:
  - *Test sets are all that we really have* for evaluating generalization.
  - Seldom possess a true test set; often, someone else has already used it.
  - Even with a true test set, evaluating subsequent models can be frustrating and unreliable.
- **Limitations of Test Sets**:
  - Only provide *post hoc* evidence of generalization.
  - Do not provide *a priori* reasons to expect generalization.
- **Appeal of Statistical Learning Theory**:
  - Aims to elucidate principles explaining why/when models trained on empirical data generalize to unseen data.
  - Primary aim: bound the generalization gap, relating model properties to dataset size.



- **Empirical vs. True Error**:
  - Learning theorists seek to bound the difference between the *empirical error* on training set and the *true error* on the population. $\epsilon_{\mathcal{S}}(f_{\mathcal{S}})$ $\mathcal{S}$ $\epsilon(f_{\mathcal{S}})$
  - Different from evaluation of a fixed classifier on a separate test set.
- **Challenges with Classifier Collections**:
  - Easy to estimate error for a single fixed classifier.
  - Difficult to estimate error accurately when considering collections of classifiers.
  - High probability of at least one classifier in the collection having a misleadingly low error.
- **Function Class Considerations**:
  - When choosing a classifier from a set of functions , there's a risk of selecting one with a grossly underestimated population error. $\mathcal{F}$
  - Linear models with continuously valued parameters typically involve choosing from an infinite class of functions ().

$$|\mathcal{F}| = \infty$$

# Statistical Learning Theory (Continued)

- **Uniform Convergence**:
  - Ambitious solution: develop tools for proving uniform convergence.
  - Goal: Empirical error rate for every classifier in the class $\mathcal{F}$ converges to its true error rate with high probability.
  - Theoretical principle: With probability at least $1 - \delta$, no classifier's error rate $\epsilon(f)$ will be misestimated by more than some small amount $\alpha$.
- **Model Class Flexibility**:
  - Not all model classes $\mathcal{F}$ can achieve uniform convergence.
  - Example: Class of memorizers (always achieve empirical error $0$ but fail on the population).
  - Tradeoff: Flexible (high variance) models fit training data well but risk overfitting. Rigid (high bias) models generalize well but risk underfitting.

- **Central Question**:
  - Quantify where a model sits on the bias-variance spectrum.
  - Provide guarantees on model performance.
- **Vapnik-Chervonenkis (VC) Dimension**:
  - Series of seminal papers by Vapnik and Chervonenkis extended theory on convergence of relative frequencies.
  - VC dimension measures the complexity (flexibility) of a model class.
  - Key result: Bound on the difference between empirical error and population error as a function of VC dimension and number of samples:

$$P\left(R[p, f] - R_{\text{emp}}[\mathbf{X}, \mathbf{Y}, f] < \alpha\right) \geq 1 - \delta \text{ for } \alpha \geq$$

# Statistical Learning Theory (Continued)

- **Key Parameters**:
  - : Probability that the bound is violated.
  - : Upper bound on the generalization gap.
  - : Dataset size.
  - : Constant depending on the scale of the loss.
- **Using the Bound**:
  - Plug in desired values of and to determine the required number of samples.
- **VC Dimension**:
  - Measures the largest number of data points for which any arbitrary (binary) labeling can be assigned and find a model in the class that agrees with that labeling.
  - Example: Linear models on -dimensional inputs have VC dimension .
    - A line can assign any possible labeling to three points in two dimensions, but not to four.

- **Practical Considerations**:
  - The theory can be overly pessimistic for complex models.
  - Obtaining the guarantee typically requires more examples than actually needed to achieve the desired error rate.
  - Fixing the model class and , the error rate decays with the usual rate.
- **Generalization Gap**:
  - While we may not do better in terms of , varying the model class can present a pessimistic picture of the generalization gap.

$\delta$ $\mathcal{O}(1/\sqrt{n})$ $n$

# Summary

- **Model Evaluation**:
  - Use a test set of previously unseen data for evaluation.
  - Test set evaluations provide an unbiased estimate of true error.
  - Converge at the $\mathcal{O}(1/\sqrt{n})$ rate as the test set grows.
  - Provide approximate confidence intervals based on asymptotic distributions or conservative finite sample guarantees.
- **Challenges with Test Sets**:
  - Often not true test sets (used repeatedly by multiple researchers).
  - Multiple evaluations on the same test set make controlling for false discovery difficult.
  - Significance of the problem depends on holdout set size and usage.
  - Good practice: curate real test sets (or multiple) and use them conservatively.

- **Statistical Learning Theory**:
  - Developed methods for guaranteeing uniform convergence over a model class.
  - Uniform convergence ensures that every model's empirical error converges to its true error simultaneously.
  - Allows for choosing the model that minimizes training error with confidence in its generalization.
- **VC Dimension**:
  - Introduced by Vladimir Vapnik and Alexey Chervonenkis.
  - Measures the complexity of a model class.
  - Provides uniform convergence results for all models in a VC class.
  - Training errors for all models in the class grow closer to their true errors at $\mathcal{O}(1/\sqrt{n})$ rates.

- **Alternative Complexity Measures**:
  - Numerous measures proposed for generalization guarantees (see :citet:boucheron2005theory for detailed discussion).
  - Useful in statistical theory but less effective for explaining deep neural networks' generalization.
- **Deep Neural Networks**:
  - Often have millions of parameters and can assign random labels to large datasets.
  - Generalize well on practical problems, often better when larger and deeper, despite larger VC dimensions.
  - Next chapter will revisit generalization in the context of deep learning.

# Exercises

1. If we wish to estimate the error of a fixed model $f$ to within $0.0001$ with probability greater than 99.9%, how many samples do we need?
2. Suppose that somebody else possesses a labeled test set $\mathcal{D}$ and only makes available the unlabeled inputs (features). Now suppose that you can only access the test set labels by running a model $f$ (no restrictions placed on the model class) on each of the unlabeled inputs and receiving the corresponding error . How many models would you need to evaluate before you leak the entire test set and thus could appear to have error , regardless of your true error? $\text{err}(f)$
3. What is the VC dimension of the class of $0$-order polynomials?
4. What is the VC dimension of axis-aligned $5^{\text{th}}$-rectangles on two-dimensional data?

# Environment and Distribution Shift

:label:`sec_environment-and-distribution-shift`

- **Context**:
  - Previous sections focused on fitting models to datasets.
  - Did not consider where data originates or the ultimate use of model outputs.
- **Common Pitfall**:
  - Rushing to develop models without considering data origins and usage.
  - Leads to failed deployments when data distribution shifts.
- **Distribution Shift Issues**:
  - Models perform well on test sets but fail in deployment.
  - Deployment can perturb data distribution.
  - Example: Loan repayment model using footwear as a feature might cause behavior change without improving credit-worthiness.

- **Critical Considerations**:
  - Decision-making based on model outputs can alter the environment.
  - Awareness of such impacts is crucial for responsible use of machine learning.
- **Mitigating Distribution Shift**:
  - Expose common concerns and stimulate critical thinking.
  - Detect situations early and mitigate damage.
  - Use machine learning responsibly.
- **Solutions**:
  - Simple: Ask for the “right” data.
  - Technically difficult: Implement reinforcement learning systems.
  - Ethical: Address philosophical questions about algorithm application.

# Types of Distribution Shift

- **Focus**:
  - Passive prediction setting.
  - Ways data distributions might shift.
  - Salvaging model performance.
- **Classic Setup**:
  - Training data sampled from distribution .
  - Test data drawn from a different distribution .
  - Learning a robust classifier is impossible without assumptions on how and relate.

$p_S(\mathbf{x}, y)$

$p_T(\mathbf{x}, y)$

$p_S$ $p_T$

- **Example**:
  - Binary classification (dogs vs. cats).
  - If $p_S(\mathbf{x}) = p_T(\mathbf{x})$ but $p_S(y \mid \mathbf{x}) = 1 - p_T(y \mid \mathbf{x})$, labels are flipped without changing the input distribution.
  - Impossible to distinguish from a setting where the distribution did not change.
- **Restricted Assumptions**:
  - Under certain assumptions, algorithms can detect shifts.
  - Algorithms can sometimes adapt on the fly, improving classifier accuracy.

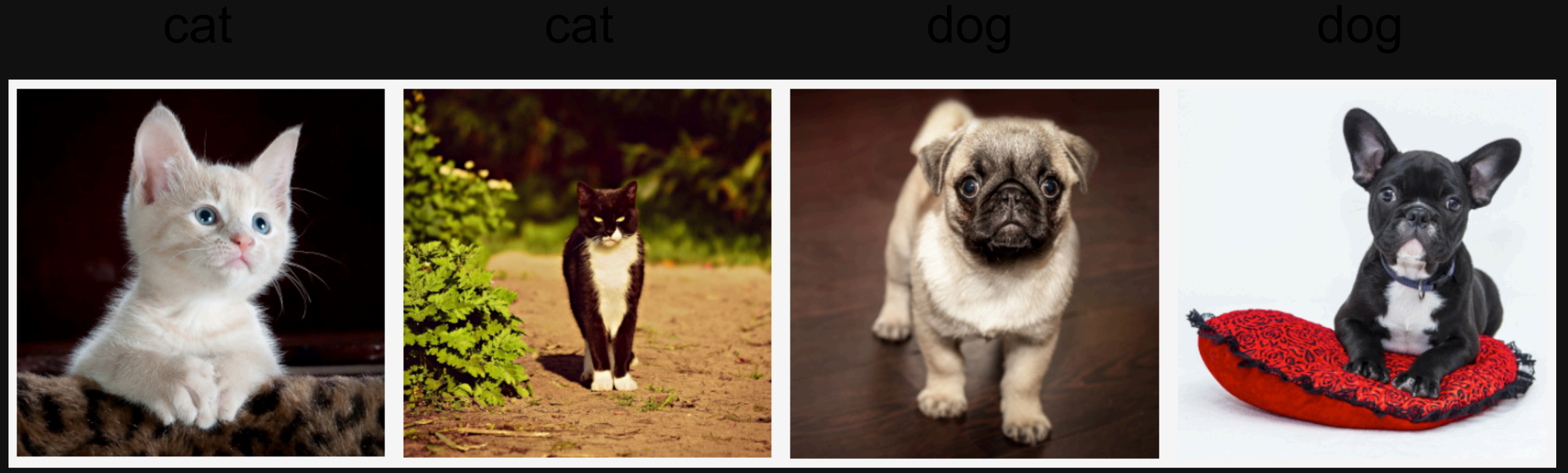
# Covariate Shift

- **Definition**:
  - Covariate shift occurs when the distribution of inputs changes over time, but the labeling function ( ) does not change.
  - The problem arises due to a shift in the distribution of the covariates (features).
- **Causality**:
  - Covariate shift is a natural assumption when causes .

$P(y \mid \mathbf{x})$

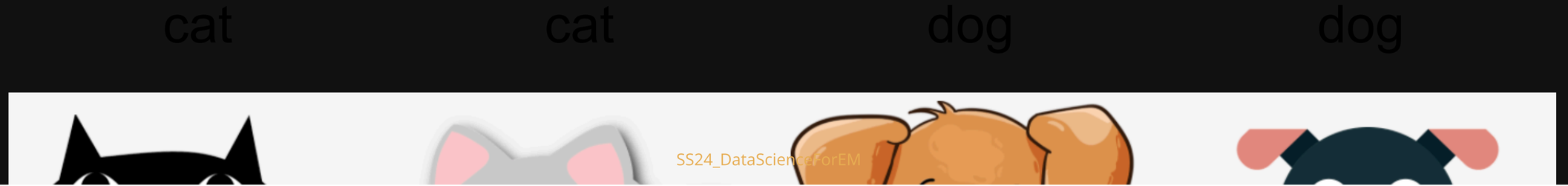
$\mathbf{x}$ $y$

- **Example**: Distinguishing cats and dogs.
  - **Training Data**:
    - Consists of photos (see :numref:`fig_cat-dog-train`).




:label:`fig_cat-dog-train`

  - **Test Data**:
    - Consists of cartoons (see :numref:`fig_cat-dog-test`).



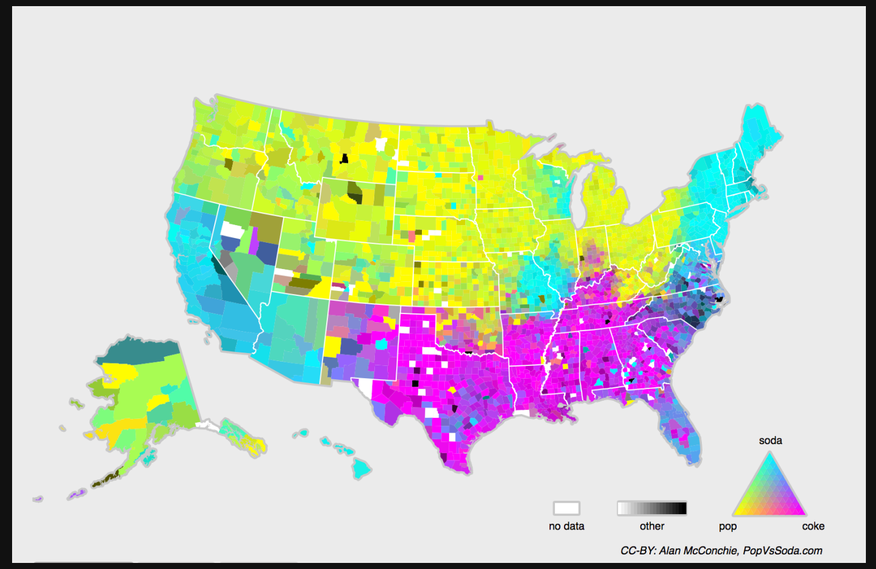
# Label Shift

- **Definition**:
  - Label shift occurs when the label marginal $P(y)$ can change, but the class-conditional distribution $P(\mathbf{x} \mid y)$ remains fixed across domains.
  - Assumption to make when $y$ causes $\mathbf{x}$.
- **Example**:
  - Predicting diagnoses given symptoms.
  - Relative prevalence of diagnoses changes over time, but diseases cause symptoms.

- **Simultaneous Occurrence**:
  - Label shift and covariate shift can hold simultaneously in degenerate cases.
  - When the label is deterministic, covariate shift assumption is satisfied even when $y$ causes $\mathbf{x}$.
- **Methodological Advantage**:
  - Methods based on label shift assumption often involve manipulating low-dimensional objects (labels).
  - This is advantageous in deep learning where inputs are high-dimensional.

# Concept Shift

- **Definition**:
  - Concept shift occurs when the definitions of labels change over time or across different contexts.
  - Example: Categories like diagnostic criteria for mental illness, fashion trends, and job titles.
- **Example**:
  - Geographic variation in names for *soft drinks* in the United States (see :numref:`fig_popvssoda`).




fig_popvssoda

- **Challenges**:
  - can vary depending on location (e.g., machine translation systems).
  - $P(y \mid \mathbf{x})$
  - Concept shift can be subtle and hard to detect.
  - Shift often occurs gradually, either temporally or geographically.

## Examples of Distribution Shift

Before delving into formalism and algorithms, let's discuss some concrete situations where covariate or concept shift might not be obvious.

# Nonstationary Distributions

- **Definition**:
  - Distribution changes slowly over time (nonstationary).
  - Model is not updated frequently enough.
- **Examples**:
  - Computational advertising model not updated with new devices (e.g., iPad launch).
  - Spam filter that fails to adapt to new spam tactics.
  - Product recommendation system recommending seasonal items out of season (e.g., Santa hats after Christmas).

# Correction of Distribution Shift

- **Overview**:
  - Training and test distributions $P(\mathbf{x}, y)$ often differ.
  - Sometimes models work despite distribution shifts; other times, principled strategies are needed.

**Empirical Risk and Risk**

:label:`subsec_empirical-risk-and-risk`

- **Model Training**:
  - Iterate over features and labels of training data $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$.
  - Update model parameters $f$ after every minibatch.
- **Empirical Risk Minimization**:
  - Minimize the loss on the training data:

:eqlabel:`eq_empirical-risk-min`

$$\underset{f}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} l(f(\mathbf{x}_i), y_i),$$

- : Loss function measuring “how bad” the prediction  is given the label .
- *Empirical risk*: Average loss over the training data to approximate the *risk*.
- *Risk*: Expectation of the loss over the entire population of data drawn from true distribution .

$l$ $f(\mathbf{x}_i)$ $y_i$ $p(\mathbf{x}, y)$

:eqlabel:`eq_true-risk`

$$E_{p(\mathbf{x}, y)}[l(f(\mathbf{x}), y)] = \int \int l(f(\mathbf{x}), y) p(\mathbf{x}, y) \, d\mathbf{x} dy.$$

- **Practical Strategy**:
  - In practice, we cannot obtain the entire population of data.
  - *Empirical risk minimization* is used to approximate minimizing the risk.

# Covariate Shift Correction

- **Goal**: Estimate $P(y \mid \mathbf{x})$ using labeled data $(\mathbf{x}_i, y_i)$ drawn from a *source distribution* $q(\mathbf{x})$ instead of the *target distribution* $p(\mathbf{x})$.
- **Dependency Assumption**: Conditional distribution does not change.: $p(y \mid \mathbf{x}) = q(y \mid \mathbf{x})$
- **Risk Correction**:
  - Use the identity to reweigh each data example:

$$\int \int l(f(\mathbf{x}), y) p(y \mid \mathbf{x}) p(\mathbf{x}) \, d\mathbf{x} dy = \int \int l(f(\mathbf{x}), y)$$

- **Weight Calculation**:
  - Reweigh each example by the ratio .
  - Train the model using *weighted empirical risk minimization*:

$$\beta_i = \frac{p(\mathbf{x}_i)}{q(\mathbf{x}_i)}$$

eq_weighted-empirical-risk-min

$$\underset{f}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} \beta_i l(f(\mathbf{x}_i), y_i).$$

- **Estimating the Ratio**:
  - Estimate the ratio $\frac{p(\mathbf{x})}{q(\mathbf{x})}$.
  - Requires samples from both distributions: true $p(\mathbf{x})$ (e.g., test data) and training set $q(\mathbf{x})$.
  - Only need features $\mathbf{x}$, not labels $y$.
- **Effective Approach**: Logistic Regression
  - Learn a classifier to distinguish between data drawn from $p(\mathbf{x})$ and data drawn from $q(\mathbf{x})$.
  - If instances are indistinguishable, they are equally likely from either distribution.
  - Well-discriminated instances should be significantly overweighted or underweighted accordingly.

# Covariate Shift Correction

- **Assumption**: Equal number of instances from distributions  and .
- **Labels**:
  - for data from .
  - for data from .
- **Probability in Mixed Dataset**:
  -
  -
- **Logistic Regression Approach**:
  -
  - Weight Calculation:

$p(\mathbf{x})q(\mathbf{x})$

$z = 1$ $p$

$z = -1$ $q$

$$P(z = 1 \mid \mathbf{x}) = \frac{p(\mathbf{x})}{p(\mathbf{x}) + q(\mathbf{x})}$$

$$\frac{P(z = 1 \mid \mathbf{x})}{P(z = -1 \mid \mathbf{x})} = \frac{p(\mathbf{x})}{q(\mathbf{x})}$$

$$P(z = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp(-h(\mathbf{x}))}$$

$$\beta_i = \exp(h(\mathbf{x}_i))$$

- **Steps to Correct Covariate Shift**:
  1. Distinguish between data from both distributions.
  2. Perform weighted empirical risk minimization using weights $\beta_i$.
- **Correction Algorithm**:
  - **Training Set**: $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
  - **Unlabeled Test Set**: $\{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$
  - **Assumption**:
    - $\mathbf{x}_i$ for $1 \leq i \leq n$ drawn from source distribution.
    - $\mathbf{u}_i$ for $1 \leq i \leq m$ drawn from target distribution.
  - **Prototypical Algorithm**:
    1. **Step 1**: Use logistic regression to estimate the weights $\beta_i$.
    2. **Step 2**: Apply weighted empirical risk minimization with the weights $\beta_i$.

1. Generate a binary-classification training set: $\{(\mathbf{x}_1, -1), \ldots, (\mathbf{x}_n, -1), (\mathbf{u}_1, 1), \ldots, (\mathbf{u}_m, 1)\}$.
2. Train a binary classifier using logistic regression to get function $h$.
3. Weigh training data using $\beta_i = \exp(h(\mathbf{x}_i))$ or better $\min(\exp(h(\mathbf{x}_i)), c)$ for some constant $c$.
4. Use weights $\beta_i$ for training on $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ in :eqref:`eq_weighted-empirical-risk-min`.

**Crucial Assumption**: - For this scheme to work, each data example in the target distribution must have a nonzero probability of occurring at training time. - If $p(\mathbf{x}) > 0$ but $q(\mathbf{x}) = 0$, the corresponding importance weight would be infinity.

# Label Shift Correction

- **Assumption**: Classification task with $k$ categories.
  - $q$ and $p$ are the source (training) and target (test) distributions, respectively.
  - Label distribution shifts: $q(y) \neq p(y)$.
  - Class-conditional distribution remains the same: $q(\mathbf{x} \mid y) = p(\mathbf{x} \mid y)$.
- **Risk Correction**:
  - Use the following identity:

$$\int \int l(f(\mathbf{x}), y) p(\mathbf{x} \mid y) p(y) \, d\mathbf{x} dy = \int \int l(f(\mathbf{x}), y)$$

- **Label Likelihood Ratios**:
  - Importance weights: $\beta_i = \frac{p(y_i)}{q(y_i)}$.
- **Estimation of Weights**:
  - Use a good model on the source distribution to get consistent estimates of these weights.
  - Inputs are high-dimensional (e.g., images), labels are simpler (e.g., categories).
- **Confusion Matrix**:
  - Compute confusion matrix $\mathbf{C}$ using the validation set.
  - $\mathbf{C}$ is a $k \times k$ matrix: columns are true labels, rows are predicted categories.
  - Each cell $c_{ij}$ is the fraction of predictions where the true label was $j$ and predicted label was $i$.
- **Estimating Target Label Distribution**:
  - Average model predictions on test data to get mean model outputs $\mu(\hat{\mathbf{y}})$.
  - $\mu(\hat{y}_i)$: fraction of predictions where the model predicted $i$.

- **Linear System**:
  - Estimate test set label distribution by solving $\mathbf{C}p(\mathbf{y}) = \mu(\hat{\mathbf{y}})$.
  - Solution: $p(\mathbf{y}) = \mathbf{C}^{-1}\mu(\hat{\mathbf{y}})$ if $\mathbf{C}$ is invertible.
- **Calculating Weights**:
  - Estimate $q(y_i)$ from source data.
  - For any training example $i$ with label $y_i$, calculate weight $\beta_i = \frac{p(y_i)}{q(y_i)}$.
  - Use weights in weighted empirical risk minimization in .eqref:eq_weighted-empirical-risk-min.

# Concept Shift Correction

- **Challenge**:
  - Concept shift is harder to correct in a principled manner.
  - Example: Shift from distinguishing cats vs. dogs to white vs. black animals may require new labels and training from scratch.
  - Extreme shifts are rare; usually, the task changes slowly over time.
- **Examples**:
  - **Computational Advertising**: New products are launched, old products become less popular. Distribution over ads and their popularity changes gradually.
  - **Traffic Cameras**: Lenses degrade gradually due to environmental wear, affecting image quality progressively.
  - **News Content**: Changes gradually with new stories appearing while most news remains unchanged.

- **Approach**:
  - Use existing network weights and perform a few update steps with new data to adapt to gradual changes, rather than training from scratch.

# A Taxonomy of Learning Problems

- **Overview**:
  - Consider other aspects of machine learning problem formulation with knowledge on dealing with distribution changes.

## Batch Learning

- **Definition**:
  - Training a model $f(\mathbf{x})$ using a batch of training features and labels $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$.
  - Model is deployed to score new data $(\mathbf{x}, y)$ drawn from the same distribution.
- **Example**:
  - Train a cat detector with pictures of cats and dogs.
  - Deploy as part of a smart catdoor system, which is then installed and never updated again.

## Online Learning

- **Definition**:
  - Data $(\mathbf{x}_i, y_i)$ arrives one sample at a time.
  - Observe $\mathbf{x}_i$, estimate $f(\mathbf{x}_i)$, then observe $y_i$ and update the model based on the incurred loss.

- **Example**:
  - Predict tomorrow's stock price, trade based on the estimate, and learn from the profit or loss at the end of the day.
- **Cycle**:

$$\text{model } f_t \longrightarrow \text{data } \mathbf{x}_t \longrightarrow \text{estimate } f_t(\mathbf{x}_t) \longrightarrow$$
$$\text{observation } y_t \longrightarrow \text{loss } l(y_t, f_t(\mathbf{x}_t)) \longrightarrow \text{model } f_{t+1}$$



## Control

- **Definition**:
  - Environment remembers previous actions, influencing current observations.
- **Examples**:
  - **PID Controller**: Adjusts heating based on previous boiler temperatures.
  - **User Behavior**: Depends on previously shown news articles.
- **Applications**:
  - Model the environment to make decisions less random.
  - Used for automatic hyperparameter tuning to improve model performance :cite:`Shao.Yao.Sun.ea.2020`.

## Considering the Environment

- **Key Distinction**:
  - Strategies for stationary environments may not work when environments adapt.
- **Example**:
  - Arbitrage opportunity disappears once exploited by a trader.
- **Algorithm Selection**:
  - Environment changes determine suitable algorithms.
  - Slow changes: Force estimates to change slowly.
  - Instantaneous but infrequent changes: Make allowances for sudden shifts.
- **Crucial Knowledge**:
  - Understanding environment dynamics is essential for dealing with concept shift.

# Summary

- **Distribution Shift**:
  - Training and test sets may come from different distributions.
  - This mismatch is known as distribution shift.
- **Risk and Empirical Risk**:
  - Risk: Expectation of the loss over the entire population drawn from the true distribution (usually unavailable).
  - Empirical Risk: Average loss over the training data, used to approximate risk.
  - Practical approach: Perform empirical risk minimization.

- **Covariate and Label Shift**:
  - These types of shifts can be detected and corrected for at test time under corresponding assumptions.
  - Ignoring these biases can lead to problematic performance during testing.
- **Environment Response**:
  - The environment may remember automated actions and respond in unexpected ways.
  - Important to consider this when building models.
  - Continuously monitor live systems to ensure models and environments do not become entangled in unanticipated ways.