Lecture 6: Generative Classifiers

Part 1: Generative Classifiers

In this lecture, we are going to look at generative classifiers and their applications.

The Iris Flowers Dataset

We will continue using the Iris dataset for classification (R. A. Fisher, 1936).

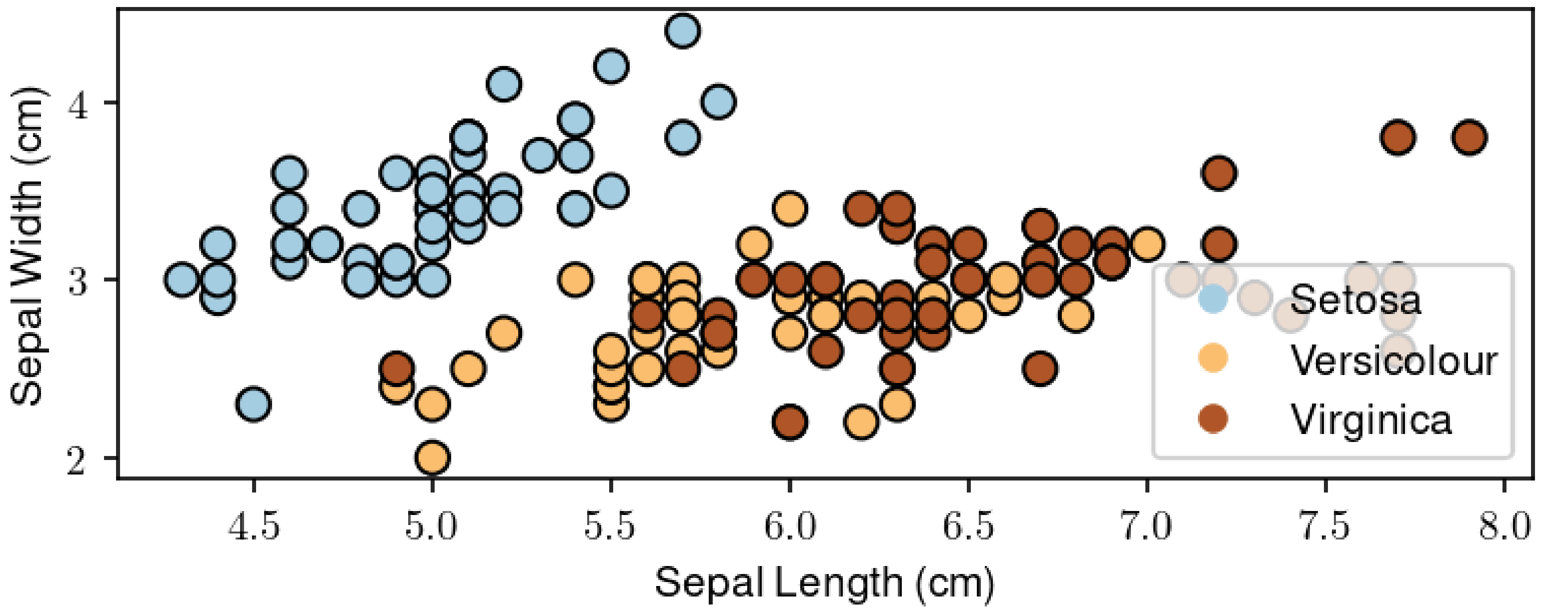
The task is to classify subspecies of Iris flowers based on their measurements:

```python
import numpy as np; np.set_printoptions(precision=2)
import pandas as pd; pd.options.display.float_format = "{:,.2f}".format
import warnings; warnings.filterwarnings('ignore')
from sklearn import datasets

iris = datasets.load_iris(as_frame=True)
iris_X, iris_y = iris.data, iris.target
pd.concat([iris_X, iris_y], axis=1).sample(5, random_state=0)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
114	5.80	2.80	5.10	2.40	2
62	6.00	2.20	4.00	1.00	1
33	5.50	4.20	1.40	0.20	0
107	7.30	2.90	6.30	1.80	2
7	5.00	3.40	1.50	0.20	0

If we only consider the first two feature columns, we can visualize the dataset in 2D.

```
# https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html
import matplotlib.pyplot as plt
plt.rcParams.update({ "figure.figsize": [6, 2], "figure.dpi": 150, "text.usetex": True, "font.family": "Helvetica"})

# create 2d version of dataset
X = iris_X.to_numpy()[:,:2]
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

# Plot also the training points
p1 = plt.scatter(X[:, 0], X[:, 1], c=iris_y, edgecolor='k', s=60, cmap=plt.cm.Paired)
plt.xlabel('Sepal Length (cm)'); plt.ylabel('Sepal Width (cm)')
plt.legend(handles=p1.legend_elements()[0], labels=['Setosa', 'Versicolour', 'Virginica'], loc='lower right');
```



Review: Supervised Learning Models

A supervised learning model can be seen as directly predicting a one-hot embedding of the target class:

$$\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\text{input}} \rightarrow \text{model } f_\theta(x) \rightarrow \underbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}}_{\text{output}}$$

In this example, the $f_\theta(x)$ predicts the second class for input x.

Review: Probabilistic Models

A probabilistic model outputs a vector of class probabilities

$$\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\text{input}} \rightarrow \text{model } P_\theta(y|x) \rightarrow \underbrace{\begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}}_{\text{output}}$$

Here, the $P_\theta(y|x)$ still predicts the second class for input x.

Example: A logistic (softmax) model

This model outputs a probability distribution

$$P_\theta(y|x) = \begin{bmatrix} P_\theta(y=0|x) \\ P_\theta(y=1|x) \end{bmatrix} = \begin{bmatrix} 1-\sigma(\theta^\top x) \\ \sigma(\theta^\top x) \end{bmatrix}$$

where $\theta^\top x$ is a linear model and

$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

is the *sigmoid* or *logistic* function.

Discriminative vs. Generative Models

Logistic regression is an example of a *discriminative* classifier.

$$P_\theta(y|x) = \begin{bmatrix} P_\theta(y=0|x) \\ P_\theta(y=1|x) \end{bmatrix}$$

- It directly transforms x into a score for each class y, e.g., via the formula $\sigma(\theta^\top x)$
- It can be interpreted as defining a *conditional* probability $P_\theta(y|x)$

Generative classifiers instead define distributions $P_\theta(x|y)$ and $P_\theta(y)$ over x, y.

Relation to text-to-image (TTI) models

- TTI models can also be written as $P(x|y)$, where x is an image (millions of dimensions) and y is a text description of it (many possibilities).
- **This lecture:** x is a flower representation (4D), y is a class (3 choices)

Stable Diffusion
DALLE 2
Midjourney
Stable Diffusion
DALLE 2
Midjourney

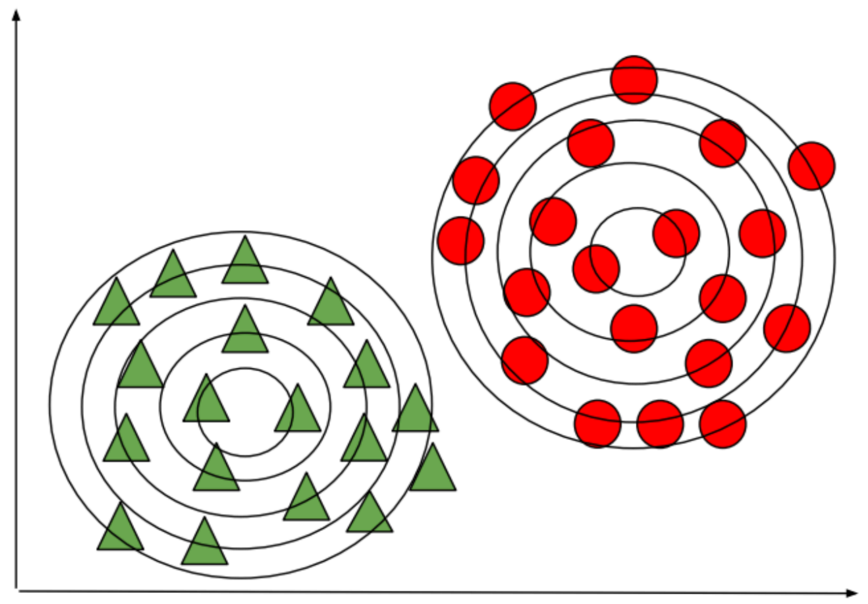
Cherry Blossom near a lake, snowing

Pyramid shaped mountain above a still lake, covered with snow

(Image source)

Generative Classifiers: Intuition

Imagine we have a dataset (e.g., flowers). We can fit a Gaussian to each class:

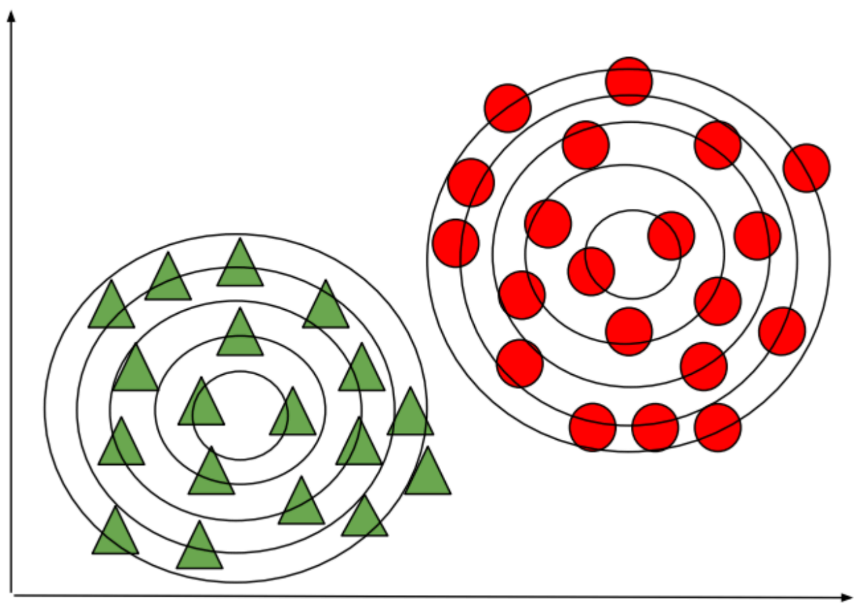


At test time, we predict the class of the Gaussian that is most likely to have generated the data.

Given a new flower x', we would compare the probabilities of three models:

$$P_\theta(x'|y=0)P_\theta(y=0)$$
$$P_\theta(x'|y=1)P_\theta(y=1)$$
$$P_\theta(x'|y=2)P_\theta(y=2)$$

We output the class that's more likely to have generated x'.



Generative Classifiers: Details

Another approach to classification is to use *generative* models.

- A generative approach first builds a model of x for each class:

$$P_\theta(x|y=k) \text{ for each class } k.$$

  $P_\theta(x|y=k)$ *scores* each x according to how well it matches class k.

- A class probability $P_\theta(y=k)$ encoding our prior beliefs. These are often just the % of each class in the data.

In the context of flower classification (e.g., Setosa vs. non-Setosa), we would fit two models on a corpus of emails x with spam/non-spam labels y:

$$P_\theta(x|y=0) \qquad \text{and} \qquad P_\theta(x|y=1)$$

- $P_\theta(x|y=1)$ *scores* each x based on how much it looks like Iris Setosa.
- $P_\theta(x|y=0)$ *scores* each x based on whether it looks like non-Setosa.

We also choose a prior $P(y)$.

Given a new x', we would compare the probabilities of both models:

$$P_\theta(x'|y=0)P_\theta(y=0) \qquad \text{vs.} \qquad P_\theta(x'|y=1)P_\theta(y=1)$$

We output the class that's more likely to have generated x'.

Part 2: Gaussian Mixture Models

Next, we will define another generative model: Gaussian mixtures.

A Generative Model for Iris Flowers

To define a generative model for Iris flowers, we need to define three probabilities:

$$P_\theta(x|y=0) \qquad P_\theta(x|y=1) \qquad P_\theta(x|y=2)$$

We also define priors $P_\theta(y=0), P_\theta(y=1), P_\theta(y=2)$.

How do we choose $P_\theta(x|y=k)$ and $P_\theta(y)$?

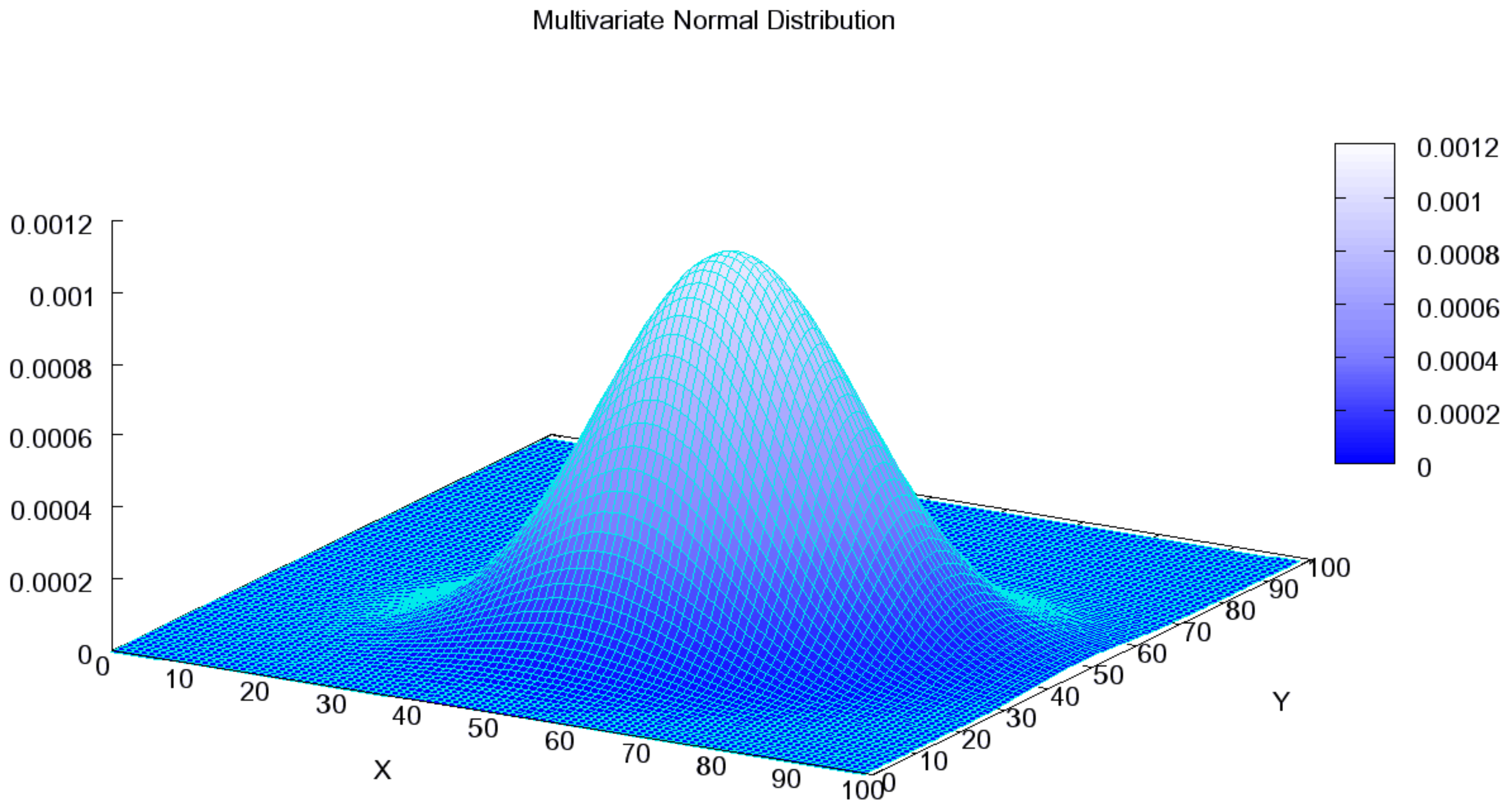
Review: Normal (Gaussian) Distribution

A multivariate normal distribution with parameters $\theta = (\mu, \Sigma)$ is a probability over a d-dimensional $x \in \mathbb{R}^d$

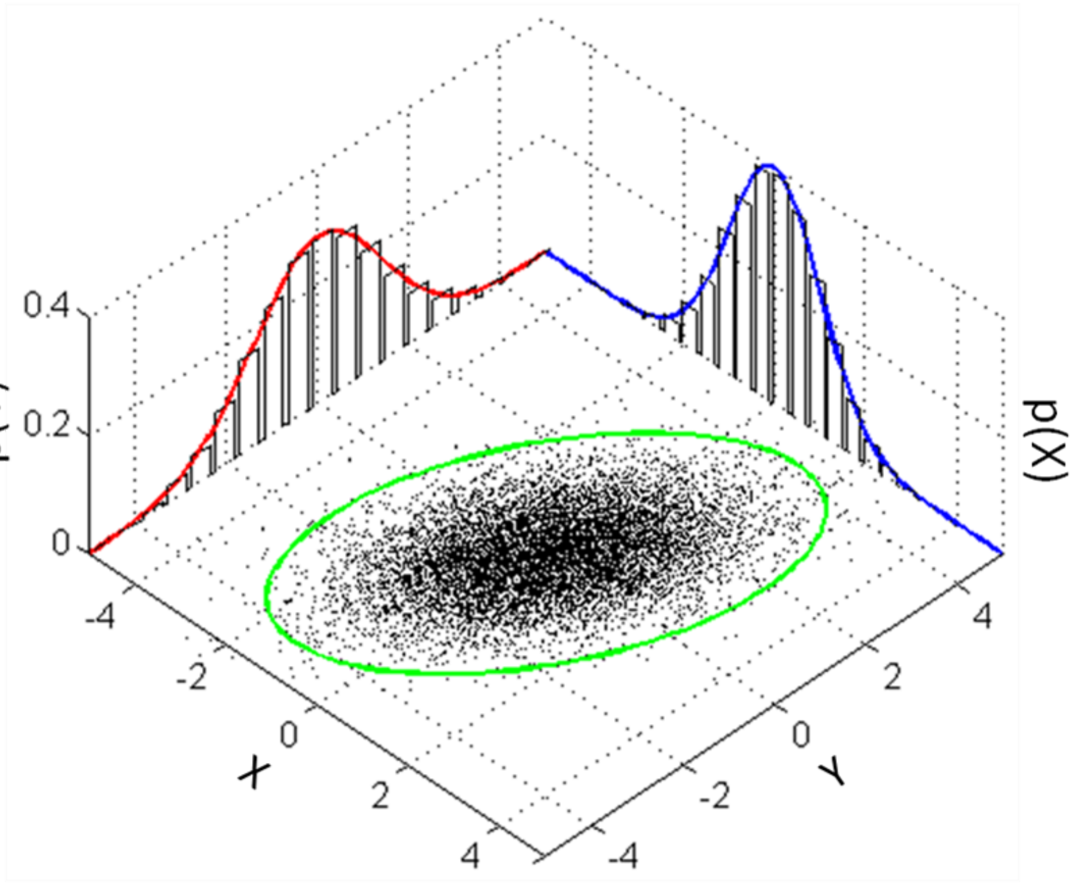
$$P_\theta(x; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right)$$

In one dimension, this reduces to $\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$.

This is what the density of a Normal distribution looks like in 2D:



This is how we can visualize it in a 2D plane:



Gaussian Mixture Model

A *Gaussian mixture* model (GMM) $P_\theta(x, y)$ is defined for *real-valued data* $x \in \mathbb{R}^d$.

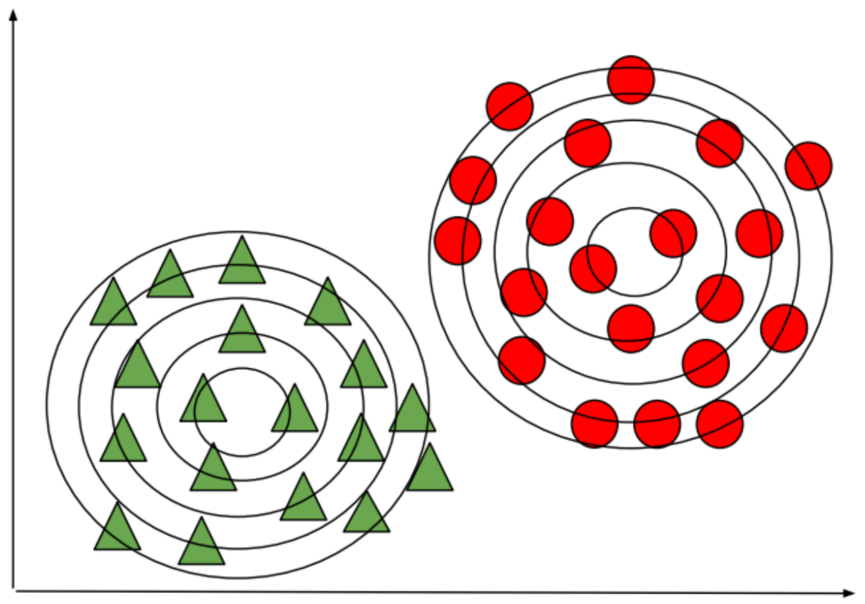
The probability of the data x for each class is a multivariate Gaussian

$$P_\theta(x|y = k) = \mathcal{N}(x; \mu_k, \Sigma_k).$$

The probability over y is Categorical: $P_\theta(y = k) = \phi_k$.

Gaussian Mixture Model: Intuition

Fitting a GMM model amounts to fitting one Gaussian per class.



At test time, we predict the class of the Gaussian that is most likely to have generated the data.

Part 3: Gaussian Discriminant Analysis

Next, we will use GMMs as the basis for a new generative classification algorithm, Gaussian Discriminant Analysis (GDA).

The Parameters of a GMM

We need to learn the parameters of two sets of distributions:

- The distribution over classes is Categorical, denoted $\text{Categorical}(\phi_1, \phi_2, \ldots, \phi_K)$. Thus, $P_\theta(y = k) = \phi_k$.
- The conditional probability $P(x \mid y = k)$ of the data under class k is a multivariate Gaussian $\mathcal{N}(x; \mu_k, \Sigma_k)$ with mean and covariance μ_k, Σ_k.

Learning the Parameters μ_k, Σ_k

- Consider the dataset of all the points x for which $y = k$.
- Goal: learn the mean and variance $\theta_k = (\mu_k, \Sigma_k)$ of a Normal that generates x
- What is the maximum likelihood μ_k, Σ_k in this case? The formulation is:

$$\arg\max_{\theta_k} P_{\theta_k}(x|y = k)$$

- For Gaussians, these are the empirical means and covariances of each class.

$$\mu_k = \frac{\sum_{i:y^{(i)}=k} x^{(i)}}{n_k}$$
$$\Sigma_k = \frac{\sum_{i:y^{(i)}=k}(x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^\top}{n_k}$$

Learning the Parameters ϕ

Next, consider learning $\vec{\phi} = (\phi_1, \phi_2, \ldots, \phi_K)$.

- We have n datapoints. Each point has a label $k \in \{1, 2, \ldots, K\}$.
- Our model is a categorical and assigns a probability ϕ_k to each outcome $k \in \{1, 2, \ldots, K\}$.
- We want to infer ϕ_k assuming our dataset is sampled from the model.

What are the maximum likelihood ϕ_k that are most likely to have generated our data?

Intuitively, the maximum likelihood class probabilities ϕ should just be the class proportions that we see in the data.

$$\phi_k = \frac{n_k}{n}$$

for each k, where $n_k = |\{i : y^{(i)} = k\}|$ is the number of training targets with class k.

Thus, the optimal ϕ_k is just the proportion of data points with class k in the training set!

Querying the Model

How do we ask the model for predictions? As discussed earlier, we can apply Bayes' rule:

$$\arg\max_y P_\theta(y|x) = \arg\max_y P_\theta(x|y)P(y).$$

Thus, we can estimate the probability of x and under each $P_\theta(x|y=k)P(y=k)$ and choose the class that explains the data best.

Algorithm: Gaussian Discriminant Analysis (GDA)

- **Type**: Supervised learning (multi-class classification)
- **Model family**: Mixtures of Gaussians.
- **Objective function**: Log-likelihood.
- **Optimizer**: Closed form solution.

Example: Iris Flower Classification

Let's see how this approach can be used in practice on the Iris dataset.

- We will learn the maximum likelihood GDA parameters
- We will compare the outputs to the true predictions.

Let's first start by computing the true parameters on our dataset.

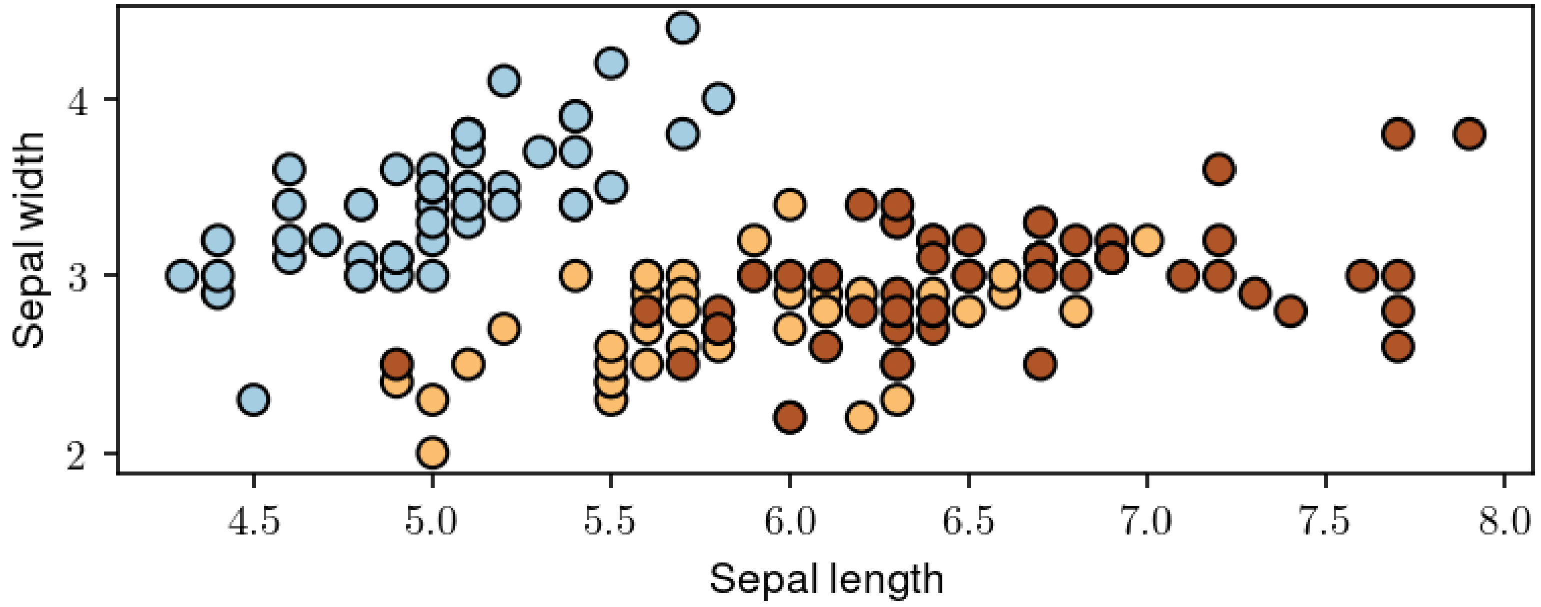
```python
# we can implement these formulas over the Iris dataset
d = 2 # number of features in our toy dataset
K = 3 # number of clases
n = X.shape[0] # size of the dataset

# these are the shapes of the parameters
mus = np.zeros([K,d])
Sigmas = np.zeros([K,d,d])
phis = np.zeros([K])

# we now compute the parameters
for k in range(3):
    X_k = X[iris_y == k]
    mus[k] = np.mean(X_k, axis=0)
    Sigmas[k] = np.cov(X_k.T)
    phis[k] = X_k.shape[0] / float(n)
```

We can look at the learned parameter values and compare them to the data

```python
print('mus:', mus, '\nphis:', phis)
```

```
mus: [[5.01 3.43]
 [5.94 2.77]
 [6.59 2.97]]
phis: [0.33 0.33 0.33]
```

```python
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired, s=50);
plt.xlabel('Sepal length'); plt.ylabel('Sepal width');
```



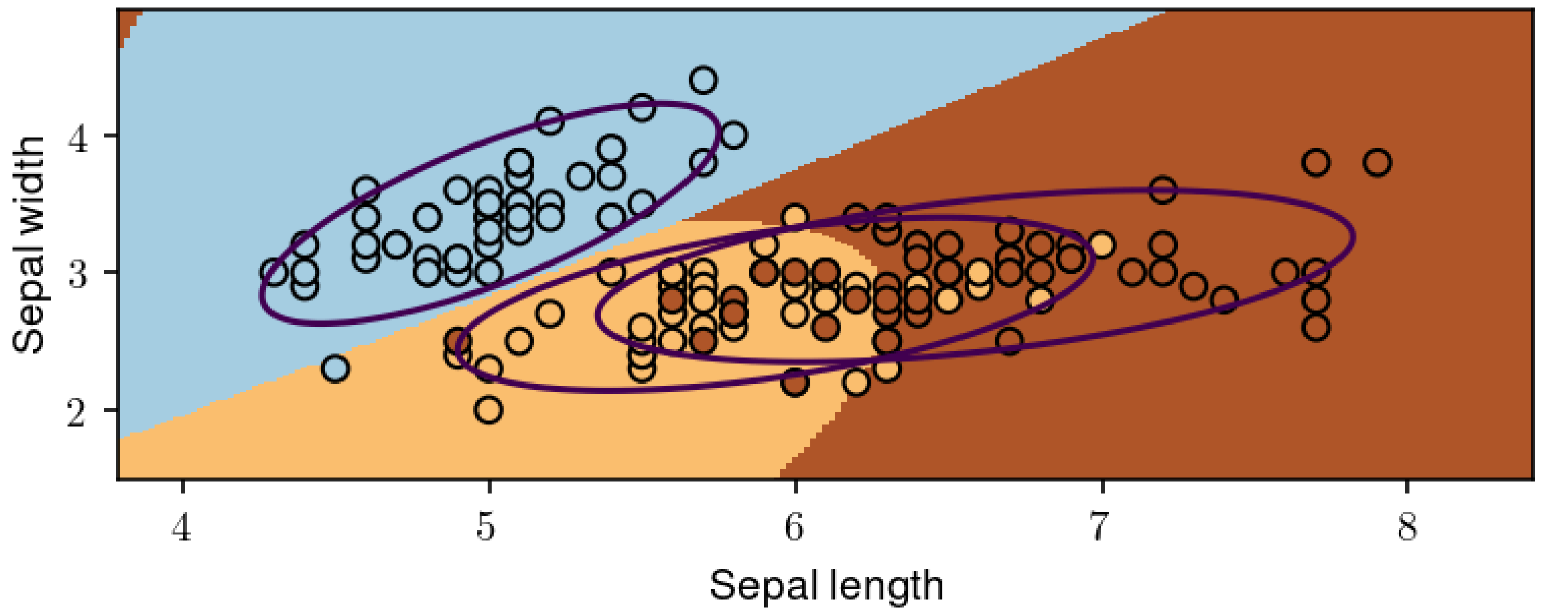
We can compute predictions using Bayes' rule.

```python
# we can implement this in numpy
def gda_predictions(x, mus, Sigmas, phis):
    """This returns class assignments and p(y|x) under the GDA model.

    We compute \arg\max_y p(y|x) as \arg\max_y p(x|y)p(y)
    """
    # adjust shapes
    n, d = x.shape
    x = np.reshape(x, (1, n, d, 1))
    mus = np.reshape(mus, (K, 1, d, 1))
    Sigmas = np.reshape(Sigmas, (K, 1, d, d))

    # compute probabilities
    py = np.tile(phis.reshape((K,1)), (1,n)).reshape([K,n,1,1])
    pxy = (
        np.sqrt(np.abs((2*np.pi)**d*np.linalg.det(Sigmas))).reshape((K,1,1,1))
        * -.5*np.exp(
            np.matmul(np.matmul((x-mus).transpose([0,1,3,2]), np.linalg.inv(Sigmas)), x-mus)
        )
    )
    pyx = pxy * py
    return pyx.argmax(axis=0).flatten(), pyx.reshape([K,n])

idx, pyx = gda_predictions(X, mus, Sigmas, phis)
print(idx)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 2 2 2 1 2 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1
 2 2 2 2 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 1 2 2 2 1 2 2 2 1 2 2 2 2 2
 2 1]
```

We visualize the decision boundaries like we did earlier.

```
Z, pyx = gda_predictions(np.c_[xx.ravel(), yy.ravel()], mus, Sigmas, phis)
logpy = np.log(-1./3*pyx)

Z = Z.reshape(xx.shape)
contours = np.zeros([K, xx.shape[0], xx.shape[1]])
for k in range(K): contours[k] = logpy[k].reshape(xx.shape)
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)
for k in range(K): plt.contour(xx, yy, contours[k], levels=np.logspace(0, 1, 1))

plt.scatter(X[:, 0], X[:, 1], c=iris_y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length'); plt.ylabel('Sepal width');
```



Special Cases of GDA

Many important generative algorithms are special cases of GDA:

- **Linear discriminant analysis (LDA):** all the covariance matrices Σ_k take the same value.
- **Gaussian Naive Bayes:** all the covariance matrices Σ_k are diagonal.
- **Quadratic discriminant analysis (QDA):** another term for GDA (because the decision boundary is quadratic).

Part 4: Discriminative vs. Generative Classifiers

We conclude our lecture on generative classifiers by revisiting the question of how they compare to discriminative classifiers.

Linear Discriminant Analysis

When the covariances Σ_k in GDA are equal, we have an algorithm called Linear Discriminant Analysis or LDA.

The probability of the data x for each class is a multivariate Gaussian with the same covariance Σ.

$$P_\theta(x|y=k) = \mathcal{N}(x; \mu_k, \Sigma).$$

The probability over y is Categorical: $P_\theta(y=k) = \phi_k$.

Let's try this algorithm on the Iris flower dataset.

We compute the model parameters similarly to how we did for GDA.

```python
# we can implement these formulas over the Iris dataset
d = 2 # number of features in our toy dataset
K = 3 # number of clases
n = X.shape[0] # size of the dataset

# these are the shapes of the parameters
mus = np.zeros([K,d])
Sigmas = np.zeros([K,d,d])
phis = np.zeros([K])

# we now compute the parameters
for k in range(3):
    X_k = X[iris_y == k]
    mus[k] = np.mean(X_k, axis=0)
    Sigmas[k] = np.cov(X.T) # this is now X.T instead of X_k.T
    phis[k] = X_k.shape[0] / float(n)

# print out the means
print(mus)
```

```
[[5.01 3.43]
 [5.94 2.77]
 [6.59 2.97]]
```

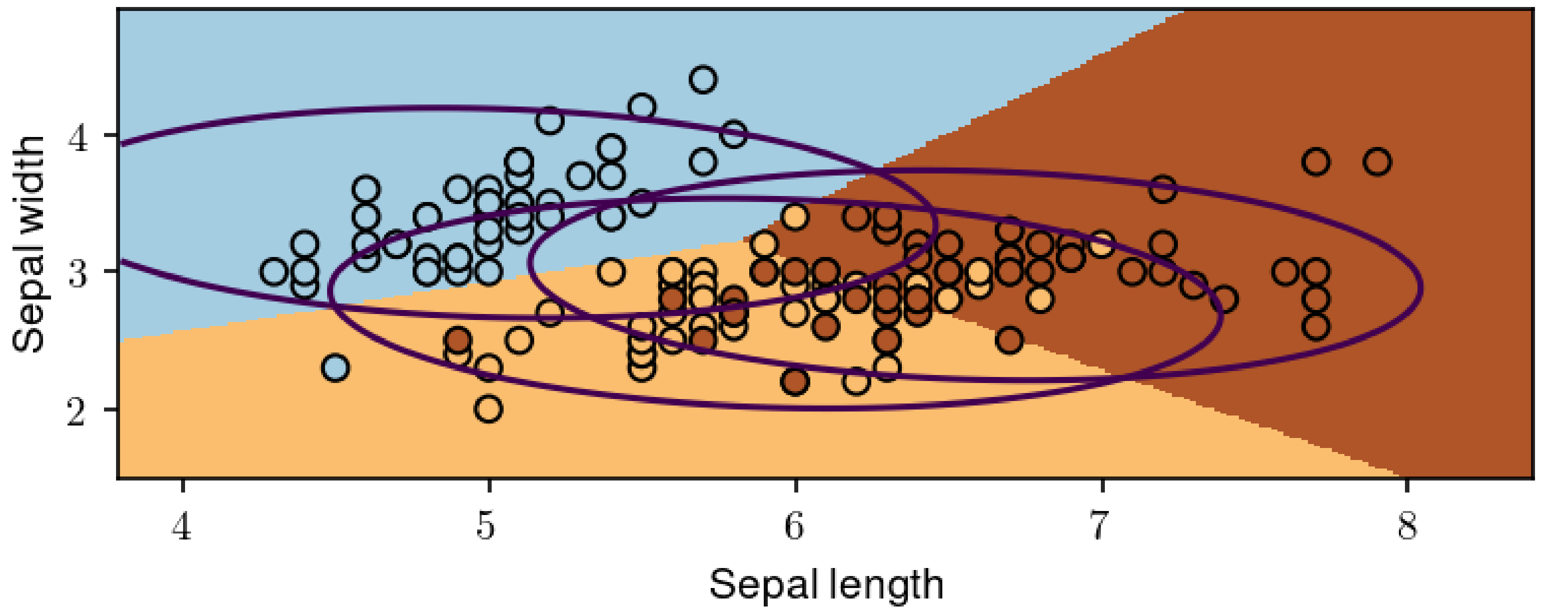
We can compute predictions using Bayes' rule.

```python
# we can implement this in numpy
def gda_predictions(x, mus, Sigmas, phis):
    """This returns class assignments and p(y|x) under the GDA model.

    We compute \arg\max_y p(y|x) as \arg\max_y p(x|y)p(y)
    """
    # adjust shapes
    n, d = x.shape
    x = np.reshape(x, (1, n, d, 1))
    mus = np.reshape(mus, (K, 1, d, 1))
    Sigmas = np.reshape(Sigmas, (K, 1, d, d))

    # compute probabilities
    py = np.tile(phis.reshape((K,1)), (1,n)).reshape([K,n,1,1])
    pxy = (
        np.sqrt(np.abs((2*np.pi)**d*np.linalg.det(Sigmas))).reshape((K,1,1,1))
        * -.5*np.exp(
            np.matmul(np.matmul((x-mus).transpose([0,1,3,2]), np.linalg.inv(Sigmas)), x-mus)
        )
    )
    pyx = pxy * py
    return pyx.argmax(axis=0).flatten(), pyx.reshape([K,n])

idx, pyx = gda_predictions(X, mus, Sigmas, phis)
print(idx)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 2 2 2 1 2 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1
 2 2 2 2 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 1 2 2
 1 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 1 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
```

We visualize predictions like we did earlier.

```
Z, pyx = gda_predictions(np.c_[xx.ravel(), yy.ravel()], mus, Sigmas, phis)
logpy = np.log(-1./3*pyx)

Z = Z.reshape(xx.shape)
contours = np.zeros([K, xx.shape[0], xx.shape[1]])
for k in range(K):
    contours[k] = logpy[k].reshape(xx.shape)
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)
for k in range(K):
    plt.contour(xx, yy, contours[k], levels=np.logspace(0, 1, 1))

plt.scatter(X[:, 0], X[:, 1], c=iris_y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length'); plt.ylabel('Sepal width');
```



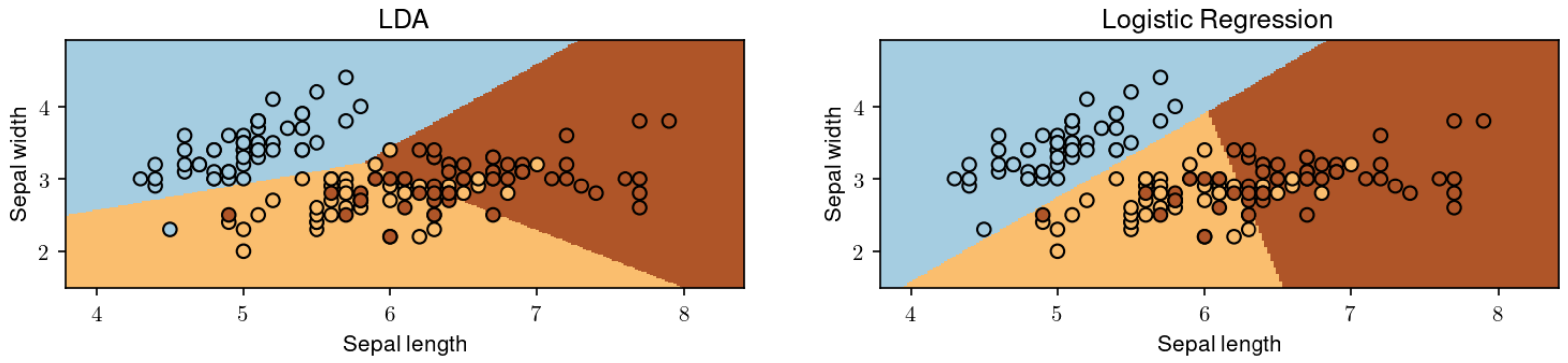
Linear Discriminant Analysis outputs decision boundaries that are linear, just like Logistic/Softmax Regression.

Softmax or Logistic regression also produce linear boundaries. In fact, both types of algorithms make use of the same model class.

Are the algorithms equivalent?

No! They assume the same model class $\mathcal{M}$, they use a different objective J to select a model in $\mathcal{M}$.

```
nrow, ncol = 1, 2
fig, axs = plt.subplots(nrow, ncol, figsize=(6*ncol, 2*nrow))
axs[0].pcolormesh(xx, yy, Z, cmap=plt.cm.Paired); axs[0].set_title('LDA')
axs[1].pcolormesh(xx, yy, Z_lr, cmap=plt.cm.Paired); axs[1].set_title('Logistic Regression')

for ax in axs:
    ax.scatter(X[:, 0], X[:, 1], c=iris_y, edgecolors='k', cmap=plt.cm.Paired)
    ax.set_xlabel('Sepal length'); ax.set_ylabel('Sepal width')
```



LDA vs. Logistic Regression

What are the differences between LDA/NB and logistic regression?

- Bernoulli Naive Bayes or LDA assumes a logistic form for $P(y|x)$. But the converse is not true: logistic regression does not assume a NB or LDA model for $P(x,y)$.
- Generative models make stronger modeling assumptions. If these assumptions hold true, the generative models will perform better.
- But if they don't, logistic regression will be more robust to outliers and model misspecification, and achieve higher accuracy.

Discriminative Approaches

Discriminative algorithms are deservingly very popular.

- Most state-of-the-art algorithms for classification are discriminative (including neural nets, boosting, SVMs, etc.)
- They are often more accurate because they make fewer modeling assumptions.

Other Useful Features of Generative Models

Generative models can also do things that discriminative models can't do.

- **Generation**: we can sample $x \sim p(x|y)$ to generate new data (images, audio).
- **Missing value imputation**: if x_j is missing, we infer it using $p(x|y)$.
- **Outlier detection**: we may detect via $p(x')$ if x' is an outlier.
- **Scalability**: Simple formulas for maximum likelihood parameters.

Part 5 (Bonus): Maximum Likelihood Learning and GDA

Here, we look at the formal reason for how to derive the learning algorithm for GDA.

Review: Maximum Likelihood Learning

We can learn a generative model $P_\theta(x, y)$ by maximizing the *maximum likelihood*:

$$\max_\theta \sum_{i=1}^n \log P_\theta(x^{(i)}, y^{(i)}).$$

This seeks to find parameters θ such that the model assigns high probability to the training data.

Let's use maximum likelihood to fit a Gaussian mixture model. Note that model parameters θ are the union of the parameters of each sub-model:

$$\theta = (\mu_1, \Sigma_1, \phi_1, \ldots, \mu_K, \Sigma_K, \phi_K).$$

Optimizing the Log Likelihood

Given a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid i = 1, 2, \ldots, n\}$, we want to optimize the log-likelihood $\ell(\theta)$:

$$\ell(\theta) = \sum_{i=1}^{n} \log P_\theta(x^{(i)}, y^{(i)}) = \sum_{i=1}^{n} \log P_\theta(x^{(i)}|y^{(i)}) + \sum_{i=1}^{n} \log P_\theta(y^{(i)})$$

$$= \sum_{k=1}^{K} \underbrace{\sum_{i:y^{(i)}=k} \log P(x^{(i)}|y^{(i)}; \mu_k, \Sigma_k)}_{\text{all the terms that involve } \mu_k, \Sigma_k} + \underbrace{\sum_{i=1}^{n} \log P(y^{(i)}; \vec{\phi})}_{\text{all the terms that involve } \vec{\phi}} .$$

Each μ_k, Σ_k for $k = 1, 2, \ldots, K$ is found in only the following terms:

$$\begin{aligned}
\max_{\mu_k, \Sigma_k} \sum_{i=1}^{n} \log P_\theta(x^{(i)}, y^{(i)}) &= \max_{\mu_k, \Sigma_k} \sum_{l=1}^{K} \sum_{i : y^{(i)} = l} \log P_\theta(x^{(i)} | y^{(i)}; \mu_l, \Sigma_l) \\
&= \max_{\mu_k, \Sigma_k} \sum_{i : y^{(i)} = k} \log P_\theta(x^{(i)} | y^{(i)}; \mu_k, \Sigma_k).
\end{aligned}$$

Thus, optimization over μ_k, Σ_k can be carried out independently of all the other parameters by just looking at these terms.

Similarly, optimizing for $\vec{\phi} = (\phi_1, \phi_2, \ldots, \phi_K)$ only involves a few terms:

$$\max_{\vec{\phi}} \sum_{i=1}^{n} \log P_\theta(x^{(i)}, y^{(i)}; \theta) = \max_{\vec{\phi}} \ \sum_{i=1}^{n} \log P_\theta(y^{(i)}; \vec{\phi}).$$

Learning the Parameters ϕ

Let's first consider the optimization over $\vec{\phi} = (\phi_1, \phi_2, \ldots, \phi_K)$.

$$\max_{\vec{\phi}} \sum_{i=1}^{n} \log P_\theta(y = y^{(i)}; \vec{\phi}).$$

- We have n datapoints. Each point has a label $k \in \{1, 2, \ldots, K\}$.
- Our model is a categorical and assigns a probability ϕ_k to each outcome $k \in \{1, 2, \ldots, K\}$.
- We want to infer ϕ_k assuming our dataset is sampled from the model.

What are the maximum likelihood ϕ_k that are most likely to have generated our data?

Intuitively, the maximum likelihood class probabilities ϕ should just be the class proportions that we see in the data.

Let's calculate this formally. Our objective $J(\vec{\phi})$ equals

$$\begin{aligned} J(\vec{\phi}) &= \sum_{i=1}^{n} \log P_\theta(y^{(i)}; \vec{\phi}) \\ &= \sum_{i=1}^{n} \log \phi_{y^{(i)}} - n \cdot \log \sum_{k=1}^{K} \phi_k \\ &= \sum_{k=1}^{K} \sum_{i : y^{(i)} = k} \log \phi_k - n \cdot \log \sum_{k=1}^{K} \phi_k \end{aligned}$$

Taking the derivative and setting it to zero, we obtain the following necessary condition

$$\frac{\phi_k}{\sum_l \phi_l} = \frac{n_k}{n}$$

for each k, where $n_k = |\{i : y^{(i)} = k\}|$ is the number of training targets with class k.

Thus, the optimal ϕ_k is just the proportion of data points with class k in the training set!

Learning the Parameters μ_k, Σ_k

Next, let's look at the maximum likelihood term

$$\max_{\mu_k, \Sigma_k} \sum_{i:y^{(i)}=k} \log \mathcal{N}(x^{(i)} | \mu_k, \Sigma_k)$$

over the Gaussian parameters μ_k, Σ_k.

- Our dataset are all the points x for which $y = k$.
- We want to learn the mean and variance μ_k, Σ_k of a normal distribution that generates this data.

What is the maximum likelihood μ_k, Σ_k in this case?

Computing the derivative and setting it to zero, we obtain closed form solutions:

$$\begin{aligned}
\mu_k &= \frac{\sum_{i:y^{(i)}=k} x^{(i)}}{n_k} \\
\Sigma_k &= \frac{\sum_{i:y^{(i)}=k} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^\top}{n_k}
\end{aligned}$$

These are just the empirical means and covariances of each class.