CS 181 Spring 2021 Section 2 Notes: Probabilistic Regression, Classification

1 Probabilistic Regression (Review)

"Story" for how the data were created.

For a model parameterized by θ, we have the **likelihood** for data $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, $\mathbf{x}_i \in \mathbb{R}^m$, $y_i \in \mathbb{R}$:

$$p(D|\theta)$$

The value of θ that maximizes the likelihood is called the maximum likelihood estimate or **MLE**.

For probabilistic regression, we modeled the conditional distribution, $p(y|\mathbf{x})$, with a target value y_i Normally distributed with mean $\mathbf{w}^\top \mathbf{x}_i$ and variance σ^2.

We showed that finding parameters $\mathbf{w}$ that minimizes the negative log likelihood of labels $\mathbf{y}$ given design matrix $\mathbf{X}$ gives the same expression for the optimal parameters $\mathbf{w}^*$ as from using ordinary least squares regression.

Later we will also see a "full Bayes" approach where we also reason about priors on the parameters θ.

2 Linear Classification

2.1 Takeaways

2.1.1 Classification

Goal : Given an input vector $\mathbf{x}$, assign it to one of K discrete classes C_k.

Input space divided into **decision regions** whose boundaries are called **decision boundaries** or **decision surfaces**.

2.1.2 Binary Linear Classification

- Two classes divided by a linear separator in feature space.
- **Discriminant function** : Function that directly assigns each vector to a specific class

$$\hat{y} = \text{sign}(h(\mathbf{x}; \mathbf{w}, w_0)) = \text{sign}(\mathbf{w}^\top \mathbf{x} + w_0)$$

- $\mathbf{w}$ is orthogonal to every point on the decision surface. It determines orientation of decision boundary.

2.1.3 Perceptron

- Discriminative algorithm for binary classification with linear decision surface

- To define the loss, we use the hinge loss / rectified linear function:

$$ReLU(z) = \max\{0, z\}$$

- Perceptron Loss :

$$\begin{aligned}
\mathcal{L}(\mathbf{w}) &= \sum_{i=1}^{n} ReLU(-h(\mathbf{x}_i; \mathbf{w}, w_0) y_i) \\
&= -\sum_{i=1: y_i \neq \hat{y}_i}^{n} (\mathbf{w}^\top \mathbf{x}_i + w_0) y_i
\end{aligned}$$

- Update using stochastic gradient descent (here, for example i):

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial}{\partial \mathbf{w}} \mathcal{L}^{(i)}(\mathbf{w}) = \mathbf{w}^{(t)} + \eta y_i \mathbf{x}_i,$$

2.2 Concept Question

Why do we choose the $ReLU$ function over the $0/1$ function when formulating the loss function?

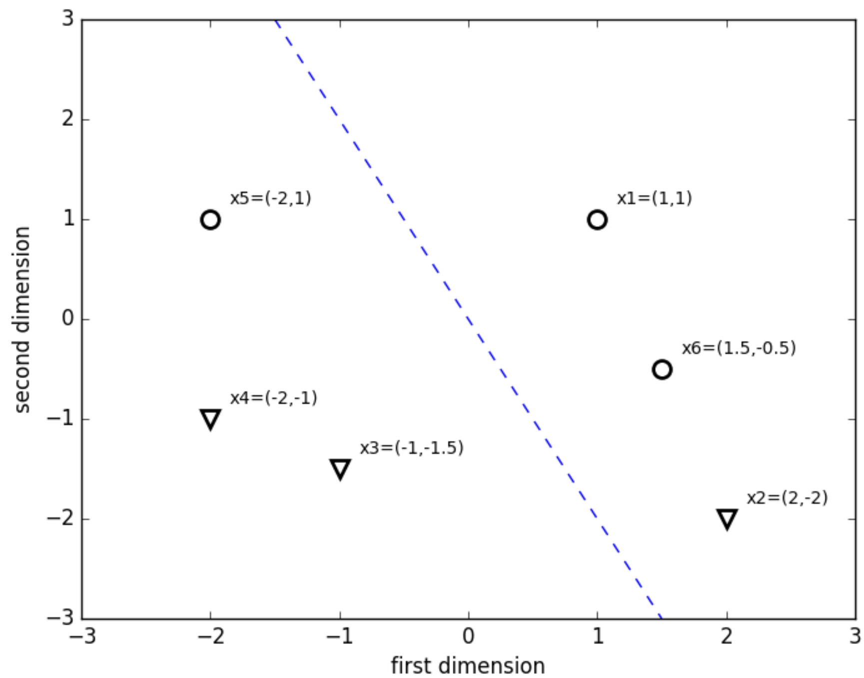
The gradient for the $0/1$ function is uninformative. We are either right or wrong.
$0/1$ is not differentiable at 0 and its derivative is 0 at all other points.

2.3 Exercise: Small Perceptron Example

Let's train a perceptron on a small data set. Consider data $\{\mathbf{x}_i\}_{i=1}^N, \mathbf{x}_i \in \mathbb{R}^2$. Let the learning rate $\eta = 0.2$ and let the weights be initialized as:

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}, w_0 = 0$$

Let the circles have $y_i = 1$ and the triangles $y_i = -1$. The data and initial separation boundary (determined by $\mathbf{w}$) is illustrated below.



Proceed by iterating over each example until there are no more classification errors. When in doubt, refer to the notes above. We know a priori that we will be able to train the classifier and have no classification errors because one can see visually that the data is linearly separable (note: as mentioned above, if the data were not so obviously linearly separable, a new basis could make it so). How many updates do you have to make? Is this surprising?

Solution:

1. Consider $\mathbf{x_1}$: $\mathbf{w}^\top \mathbf{x_1} + w_0 = 1 \cdot 1 + 0.5 \cdot 1 + 0 > 0$. This is a correct classification, so we take no action.
2. Consider $\mathbf{x_2}$: $\mathbf{w}^\top \mathbf{x_2} + w_0 = 1 \cdot 2 + 0.5 \cdot (-2) + 0 > 0$. This is an incorrect classification, so we need to update our weight parameters:

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} + (0.2)(-1)\begin{pmatrix} 2 \\ -2 \end{pmatrix} = \begin{pmatrix} 0.6 \\ 0.9 \end{pmatrix} \\ w_0 &\leftarrow w_0 + 0.2(-1) = -0.2 \end{aligned}$$

3. Consider $\mathbf{x_3}$: $\mathbf{w}^\top \mathbf{x_3} + w_0 = 0.6 \cdot (-1) + 0.9 \cdot (-1.5) + (-0.2) < 0$. This is a correct classification.
4. Consider $\mathbf{x_4}$: Check that this is a correct classification.
5. Consider $\mathbf{x_5}$: Check that this is an incorrect classification and our weight parameters are updated to:

$$\begin{aligned} \mathbf{w} &\leftarrow \begin{pmatrix} 0.2 \\ 1.1 \end{pmatrix} \\ w_0 &\leftarrow 0 \end{aligned}$$

6. Consider $\mathbf{x_6}$: Check that this is again an incorrect classification:

$$\begin{aligned} \mathbf{w} &\leftarrow \begin{pmatrix} 0.5 \\ 1 \end{pmatrix} \\ w_0 &\leftarrow 0.2 \end{aligned}$$

All data points are correctly classified now. If the data is linearly separable, we are guaranteed to converge to a solution using the perceptron algorithm in a finite number of steps.

3 Probabilistic Classification

3.1 Takeaways

3.1.1 Discriminative Model

We need to model $p(y|\mathbf{x})$.

In the binary case, we can use the **sigmoid function**:

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

and the **negative log likelihood** loss function:

$$\mathcal{L}(\theta) = -\sum_{n=1}^{N} (y_n \ln p(y_n = 1|\mathbf{x}_n;\theta) + (1-y_n)\ln p(y_n = 0|\mathbf{x}_n;\theta))$$

where θ are the parameters of the model. Note, in classification settings, the negative log likelihood loss is sometimes called the **cross-entropy loss** due being related to the cross-entropy function.

Logistic Regression: Use a linear function

$$z = \mathbf{w}^T\mathbf{x} + w_0$$

3.1.2 Generative Model

Models the joint distribution $p(\mathbf{x}, y)$. This factors into $p(\mathbf{x}|y)p(y)$.

- $p(y)$ is called the **class prior** and is always a categorical distribution, and just Bernoulli for binary classification.
  Gives an a priori probability of an observation being a certain class, without even considering the observation's features.
- $p(x|y)$ is called the **class-conditional distribution** and its form is model-specific.
  Specifies how likely an observation (set of features) is given a class.

We are interested in picking the class k that maximizes $p(y = k|\mathbf{x})$.

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})} \propto p(\mathbf{x}|y)p(y)$$

$\mathbf{x}$ can be either discrete or continuous.

3.1.3 Naive Bayes

3.1.4 Naive Bayes

Naive Bayes is one type of generative model for classification. It's "naive" because we assume that each dimension $d \in \{1, ..., D\}$ of the nth observed data point $\mathbf{x}_n$ is conditionally independent from the other dimensions given the correct class label i.e. $y_n = C_k$.

$$p(\mathbf{x}_n | y_n = C_k) = \prod_{d=1}^{D} p(x_{nd} | y_n = C_k)$$

For a concrete example, suppose we're classifying movies into demographics the movies will appeal to based on three features: (1) the decade in which the star actor/actress was born, (2) the kind of marketing (e.g. blog vs magazine vs newspaper) and (3) the country the movie is made in. For each class C_k, and each feature $d \in \{1, 2, 3\}$, we model the probability of the feature value as a Categorical distribution with probability vector π_{kd}:

$$\pi_{kd} = [\pi_{kd1}, \pi_{kd2}, ..., \pi_{kdJ}]^\top \tag{1}$$

with $\sum_{j=1}^{J} \pi_{kdj} = 1$ for all k, d. The probability of dimension d of a data point $\mathbf{x}_n$ in class C_k is

$$\begin{aligned} p(x_{nd} | y_n = C_k) &= Cat(x_{nd} | \pi_{kd}) && (2) \\ &= \prod_{j=1}^{J} \pi_{kdj}^{\mathbb{I}[x_{nd}=j]} && (3) \end{aligned}$$

Under the Naive Bayes assumption, we assume that each feature's distribution is independent from the other features' distributions given the class. This means the conditional probability of the nth film summary given the kth class can be written as :

$$\begin{aligned} p(\mathbf{x}_n | y_n = C_k) &= \prod_{d=1}^{D} p(x_{nd} | y_n = C_k) && (4) \\ &= \prod_{d=1}^{D} \prod_{j=1}^{J} \pi_{kdj}^{\mathbb{I}(x_{nd}=j)} && (5) \end{aligned}$$

3.1.5 Naive Bayes Concept Questions

How many parameters does this model have? Why do we use the "naive" assumption?

The model has $K \times D \times (J-1)$ parameters. The $J-1$ arises because each Categorical distributions' J parameters must sum to 1, removing one degree of freedom. We use the "naive" assumption in order to keep the number of parameters small.
Naive bayes requires a number of parameters linear in the number of variables (classes, features).

3.1.6 Naive Bayes Practice Problem

Let's consider just two classes. Let $p(y = 1) = \theta$ and $p(y = 0) = 1 - \theta$. For a given dataset $\{\mathbf{x}_n, y_n\}_{n=1}^N$, what are the maximum likelihood estimates of the the parameters $\theta, \{\pi_{k,d,j}\}$? For MLE, we want to find the parameters that minimize the negated log-likelihood:

$$
\begin{aligned}
L(\theta, \{\pi_{k,d,j}\}) &= -\ln p(\{\mathbf{x_n}, y_n\}_{n=1}^N | \theta, \{\pi_{k,d,j}\}) && (6)\\
&= -\ln \prod_{n=1}^N p(\mathbf{x_n}, y_n | \theta, \{\pi_{k,d,j}\}) && (7)\\
&= -\ln \prod_{n:y_n=1} p(\mathbf{x_n} | \{\pi_{1,d,j}\}) p(y_n = 1|\theta) \prod_{n:y_n=0}^N p(\mathbf{x_n} | \{\pi_{0,d,j}\}) p(y_n = 0|\theta) && (8)\\
&= -\sum_{n:y_n=1} \ln p(\mathbf{x_n} | \{\pi_{1,d,j}\}) - \sum_{n:y_n=1} \ln p(y_n = 1|\theta) && (9)\\
&\quad - \sum_{n:y_n=0} \ln p(\mathbf{x_n} | \{\pi_{0,d,j}\}) - \sum_{n:y_n=0} \ln p(y_n = 0|\theta) && (10)\\
&= -\sum_{n:y_n=1} \ln \prod_{d=1}^D \prod_{j=1}^J \pi_{1dj}^{\mathbb{I}[x_{nd}=j]} - \sum_{n:y_n=1} \ln(\theta) && (11)\\
&\quad - \sum_{n:y_n=0} \ln \prod_{d=1}^D \prod_{j=1}^J \pi_{0dj}^{\mathbb{I}[x_{nd}=j]} - \sum_{n:y_n=0} \ln(1-\theta) && (12)\\
&= -\sum_{n:y_n=1} \sum_{d=1}^D \sum_{j=1}^J \mathbb{I}[x_{nd} = j] \ln(\pi_{1dj}) - \sum_{n:y_n=1} \ln(\theta) && (13)\\
&\quad - \sum_{n:y_n=0} \sum_{d=1}^D \sum_{j=1}^J \mathbb{I}[x_{nd} = j] \ln(\pi_{0dj}) - \sum_{n:y_n=0} \ln(1-\theta) && (14)
\end{aligned}
$$

The indicator function $\mathbb{I}[\cdot]$ is useful to make sure that our likelihood only evaluates $p(\mathbf{x}, y)$ for each $\mathbf{x}$'s true class rather than both classes. This is used a lot, so make sure you see what is going on.

3.2 Exercise: Shapes of Decision Boundaries I

Consider now a generative model with $K > 2$ classes, and output label $\mathbf{y}$ encoded as a "one hot" vector of length K. We adopt class prior $p(\mathbf{y} = C_k; \boldsymbol{\pi}) = \pi_k$ for all $k \in \{1, \ldots, K\}$ (where π_k is a parameter of the prior). Let $p(\mathbf{x} \,|\, \mathbf{y} = C_k)$ denote the class-conditional density of features $\mathbf{x}$ (in this case for class C_k). Let the class-conditional probabilities be Gaussian distributions

$$p(\mathbf{x} \,|\, \mathbf{y} = C_k) = \mathcal{N}(\mathbf{x} \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \text{ for } k \in \{1, \ldots, K\}$$

We will predict the class of a new example $\mathbf{x}$ as the class with the highest conditional probability, $p(\mathbf{y} = C_k \,|\, \mathbf{x})$. Luckily, a little bird came to the window of your dorm, and claimed that you can classify an example $\mathbf{x}$ by finding the class that maximizes the following function:

$$f_k(\mathbf{x}) = \log(\pi_k) - \frac{1}{2}\log(|\boldsymbol{\Sigma}_k|) - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k).$$

Derive this formula by comparing two different classes' conditional probabilities. What can we claim about the shape of the decision boundary given this formula?

Solution:

Let's start with the conditional probability:

$$p(\mathbf{y} = C_k \,|\, \mathbf{x}) = \frac{p(\mathbf{x} \,|\, \mathbf{y} = C_k)p(\mathbf{y} = C_k)}{\sum_\ell^c p(\mathbf{x} \,|\, \mathbf{y} = C_\ell)p(\mathbf{y} = C_\ell)}$$

{We can take out the denominator since it will be the same for each class}

$$\begin{aligned} &\propto p(\mathbf{x} \,|\, \mathbf{y} = C_k)p(\mathbf{y} = C_k) \\ &= \frac{1}{(2\pi)^{\frac{m}{2}}|\boldsymbol{\Sigma}_k|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right)\pi_k \end{aligned}$$

{We can factor out constants that will be shared across classes}

$$\propto \frac{\pi_k}{|\boldsymbol{\Sigma}_k|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right)$$

{We take the logarithm, which is a monotonically increasing function and won't change the maximum across classes}

$$\propto \ln(\pi_k) - \frac{1}{2}\ln(|\boldsymbol{\Sigma}_k|) - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)$$

Now, what can we say about the shape of our decision boundary? $(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)$ gives us intuition about the shape of our decision boundary. We see it is quadratic.

For further intuition, we can look at the other two terms to see what affects the probability of an example being classified to a given class. As the prior π_k increases, we see that the probability increases, which fits our intuition. As the covariance matrix $\boldsymbol{\Sigma}_k$ increases, we see that the probability decreases, which also fits our intuition.

3.3 Exercise: Shapes of Decision Boundaries II

Let's say the little bird comes back and now tells you that every class has the same covariance matrix, and so $\mathbf{\Sigma}_\ell = \mathbf{\Sigma}'_\ell$ for all classes C_ℓ and $C_{\ell'}$. Simplify this formula down further. What can we claim about the shape of the decision boundaries now?

Solution:

Here was our original formula.

$$\ln(\pi_k) - \frac{1}{2}\ln(|\mathbf{\Sigma}_k|) - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) \tag{15}$$

Given that all of the classes have the same covariance matrix, which we will write as $\mathbf{\Sigma}$, we have:

$$\begin{aligned}
f_k(\mathbf{x}) &= \ln(\pi_k) - \frac{1}{2}\ln(|\mathbf{\Sigma}|) - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) \\
&= \ln(\pi_k) - \frac{1}{2}\ln(|\mathbf{\Sigma}|) - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T (\mathbf{\Sigma}^{-1}\mathbf{x} - \mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k) \\
&= \ln(\pi_k) - \frac{1}{2}\ln(|\mathbf{\Sigma}|) - \frac{1}{2}(\mathbf{x}^T\mathbf{\Sigma}^{-1}\mathbf{x} - \mathbf{x}^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k - \boldsymbol{\mu}^T\mathbf{\Sigma}^{-1}\mathbf{x} + \boldsymbol{\mu}^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k) \\
&= \ln(\pi_k) - \frac{1}{2}\ln(|\mathbf{\Sigma}|) - \frac{1}{2}(\mathbf{x}\mathbf{\Sigma}^{-1}\mathbf{x}^T - 2\mathbf{x}^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k + \boldsymbol{\mu}_k^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k) \\
&= \ln(\pi_k) - \frac{1}{2}\ln(|\mathbf{\Sigma}|) - \frac{1}{2}\mathbf{x}\mathbf{\Sigma}^{-1}\mathbf{x}^T + \mathbf{x}^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k
\end{aligned}$$

$\{$We can drop $-\frac{1}{2}\ln(|\mathbf{\Sigma}|)$ and $-\frac{1}{2}\mathbf{x}\mathbf{\Sigma}^{-1}\mathbf{x}^T$ since they are class independent$\}$

$$\propto \ln(\pi_k) + \mathbf{x}^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k$$

Thus, we conclude that we can adopt function

$$\tilde{f}_k(\mathbf{x}) = \mathbf{x}^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^T\Sigma^{-1}\boldsymbol{\mu}_k + \ln(\pi_k) \tag{16}$$

Looking at this formula, we can see that our decision boundaries are no longer quadratic but linear. We've proven that if the classes share the same covariance matrix, our decision boundaries will be linear!

3.4 Visualizing Decision Boundaries

If you want to better understand what these decision boundaries look like, we can visualize them! Let's consider two classes and assume x lives in 2 dimensions. We first consider the case in which the two classes have identical covariances, here defined as

$$p(x|y=1) = \mathcal{N}\Big(\mu_1 = \begin{bmatrix}1\\1\end{bmatrix}, \Sigma_1 = \begin{bmatrix}2 & 0\\0 & 1\end{bmatrix}\Big)$$
$$p(x|y=2) = \mathcal{N}\Big(\mu_2 = \begin{bmatrix}-1\\-1\end{bmatrix}, \Sigma_2 = \begin{bmatrix}2 & 0\\0 & 1\end{bmatrix}\Big)$$

We use the following code to plot the contours of both Gaussians:

[language=Python] imports import matplotlib.pyplot as plt import numpy as np import plotly.graph$_{o}bjectsasgoi$

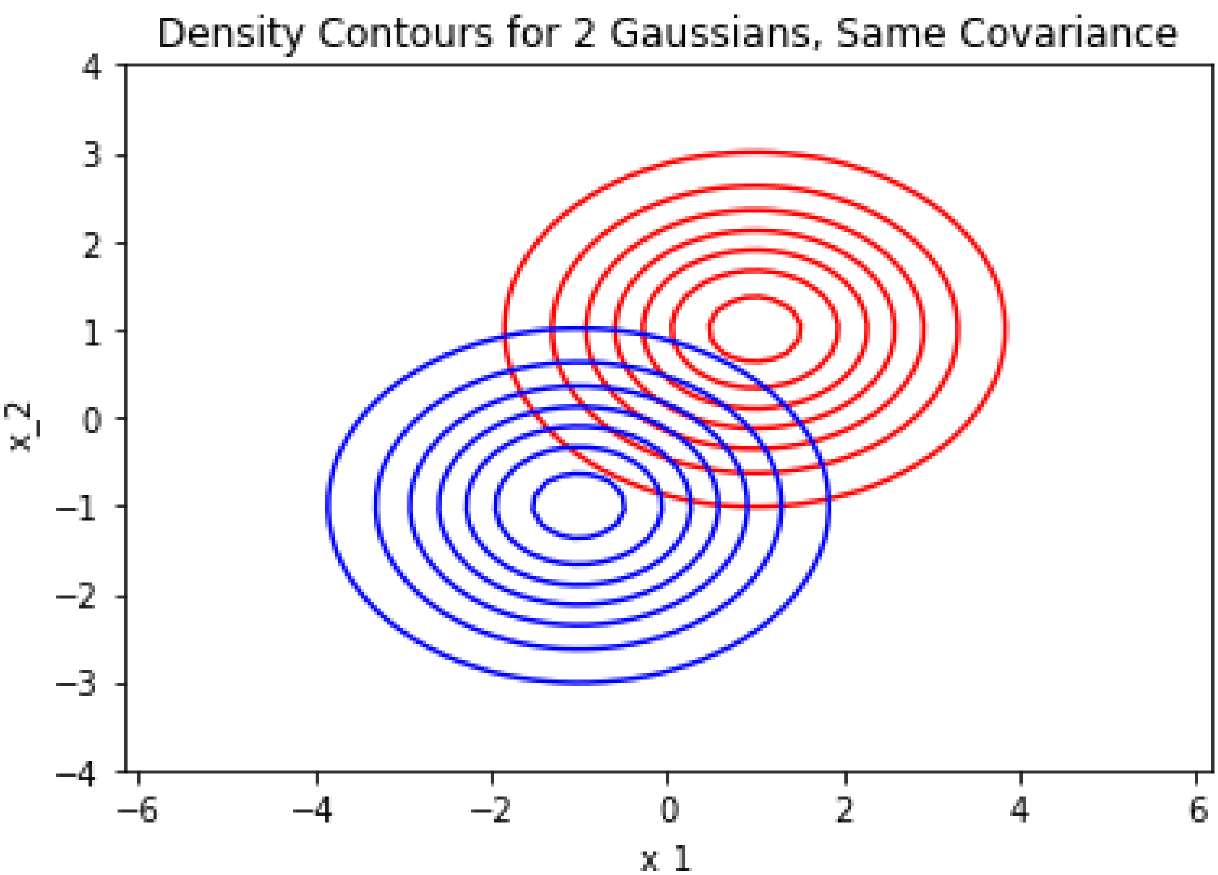
create meshgrid from -8 to 8 mesh$_{g}ranularity = 100possible_{v}als = np.linspace(-8., 8, mesh_{g}ranularity)mesh_{c}oo$
$np.meshgrid(possible_{v}als, possible_{v}als)mesh_{c}oords = np.reshape(np.stack(mesh_{c}oords), newshape =$
$(2, mesh_{g}ranularity * mesh_{g}ranularity)).T$

define means of Gaussians means = [np.array([1., 1.]), np.array([-1., -1.])]

compute densities for both Gaussians assuming equal covariances covs = [np.array([[2., 0.], [0., 1.]]), np.array([[2., 0.], [0., 1.]])] same$_{c}ov_{d}ensities = [scipy.stats.multivariate_{n}ormal.pdf(x = mesh_{c}oords, mean = mean, cov = cov)formean, covinzip(means, covs)]$

plot plt.contour(np.reshape(mesh$_{c}oords[:, 0], newshape = (mesh_{g}ranularity, mesh_{g}ranularity)), np.reshape(mes$
$, 1], newshape = (mesh_{g}ranularity, mesh_{g}ranularity)), np.reshape(same_{c}ov_{d}ensities[0], newshape =$
$(mesh_{g}ranularity, mesh_{g}ranularity)), colors =' red')plt.contour(np.reshape(mesh_{c}oords[:, 0], newshape =$
$(mesh_{g}ranularity, mesh_{g}ranularity)), np.reshape(mesh_{c}oords[:, 1], newshape = (mesh_{g}ranularity, mesh_{g}ranula$
$(mesh_{g}ranularity, mesh_{g}ranularity)), colors =' blue')plt.xlabel('x_1')plt.ylabel('x_2')plt.title('DensityContoursfor$

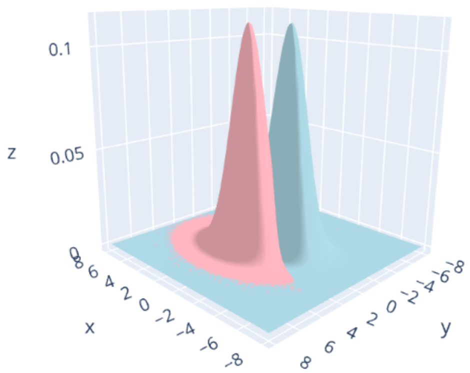
This gives us the following contours:

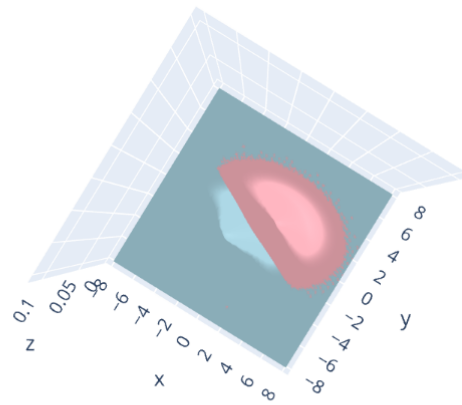


If you would prefer plotting in 3D, we can alternatively use Plotly:

[language=Python] fig = go.Figure(data=[go.Mesh3d(x=mesh$_c oords[:, 0], y = mesh_c oords[:, 1], z = same_c ov_d ensities[0], color =' lightpink'), go.Mesh3d(x = mesh_c oords[:, 0], y = mesh_c oords[:, 1], z = same_c ov_d ensities[1], color =' lightblue')]) fig.show()$

Rotating the plot (and ignoring the floating point problems on the periphery), we can see that the two Gaussians have equal density along a line:





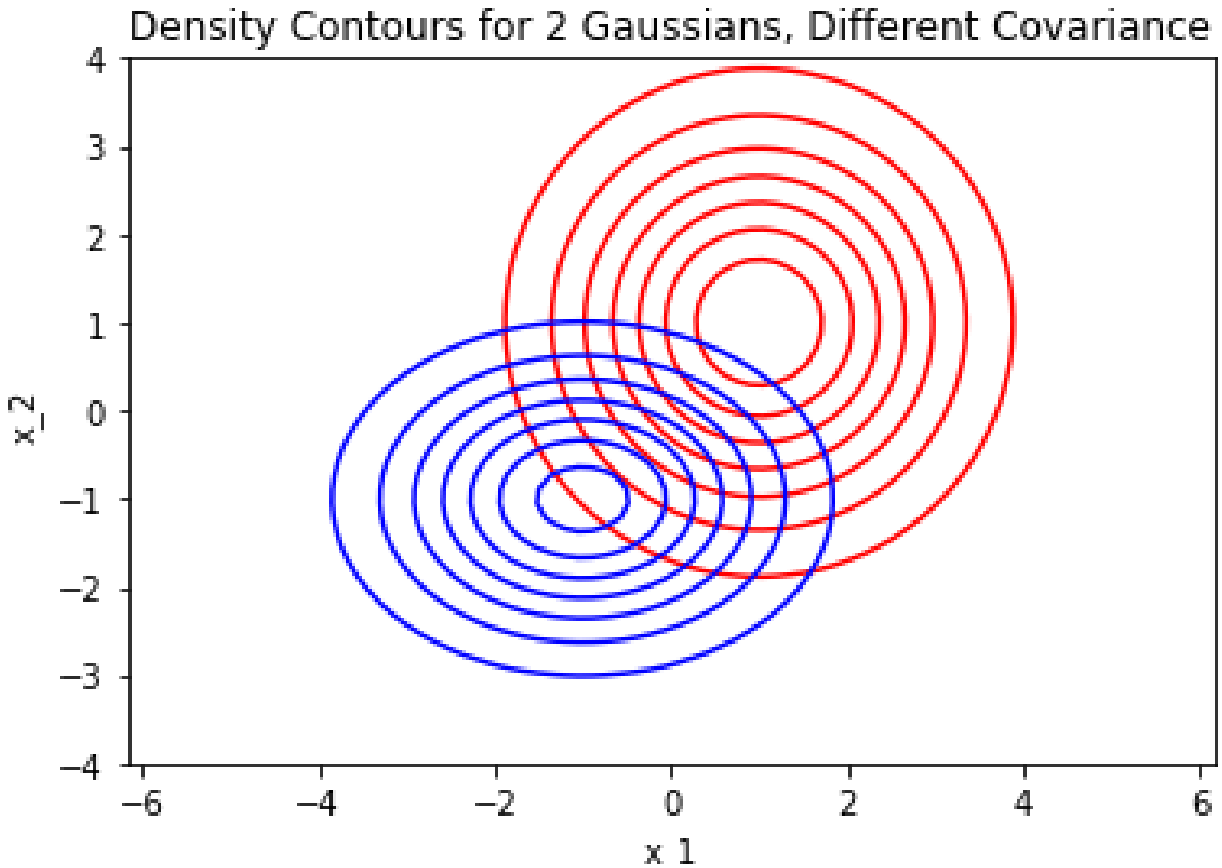
We now plot the second case, where covariances are unequal. We assume that the two Gaussians are:

$$p(x|y=1) = \mathcal{N}\Big(\mu_1 = \begin{bmatrix}1\\1\end{bmatrix}, \Sigma_1 = \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix}\Big)$$
$$p(x|y=2) = \mathcal{N}\Big(\mu_2 = \begin{bmatrix}-1\\-1\end{bmatrix}, \Sigma_2 = \begin{bmatrix}2 & 0\\0 & 1\end{bmatrix}\Big)$$

define means of Gaussians means = [np.array([1., 1.]), np.array([-1., -1.])]

compute densities for both Gaussians assuming equal covariances covs = [2.*np.eye(2), np.array([[2., 0.], [0., 1.]])] diff$_c ov_d ensities = [scipy.stats.multivariate_n ormal.pdf(x = mesh_c oords, mean = mean, cov = cov) for mean, cov in zip(means, covs)]$

The contour plot


Density Contours for 2 Gaussians, Different Covariance
x_2
x_1
4
3
2
1
0
−1
−2
−3
−4
−6
−4
−2
0
2
4
6

Rotating the plot (and ignoring the floating point problems on the periphery), we can see that the two Gaussians have equal density along a parabola:

[language=Python] fig = go.Figure(data=[go.Mesh3d(x=mesh$_{c}oords[:,0], y = mesh_{c}oords[:,1], z = diff_{c}ov_{d}ensities[0], color =' lightpink'), go.Mesh3d(x = mesh_{c}oords[:,0], y = mesh_{c}oords[:,1], z = diff_{c}ov_{d}ensities[1], color =' lightblue')]) fig.show()$