Domácí úloha SUR 2020-2021

April 10, 2021

Vítejte u domácí úlohy do SUR. V rámci úlohy Vás čeká několik cvičení, v nichž budete doplňovat poměrně malé fragmenty kódu, místo na ně je vyznačené jako `pass`, `...` nebo `None`. Nebojte se přidávat další řádky pro výpočet dílčích výsledků, ale pokud se v buňce s kódem již něco nachází, využijte/neničte to. Buňky nerušte ani nepřidávejte.

Maximálně využívejte `numpy`, nikde by se neměl objevit cyklus jdoucí přes jednotlivé příklady. Celkově Vás čeká docela dost `numpy`-gymnastiky. Seznamte se s tím, jakou roli hraje argument `axis` při různých kolapsech polí (např. v `np.sum()`). Ujistěte se, že rozumíte `np.concatenate()` a `np.stack()` pro spojování polí. A nezapomeňte, že zatímco násobení polí (`*`) pouze násobí, skalární/maticový součin (`@`) navíc i vysčítá danou dimenzi. Dále se Vám nejspíše bude hodit porozumění broadcastingu.

Před odevzdáním spusťte celý notebook načisto (Kernel -> Restart and Run all). Pokud přitom selže některý krok (např. strašlivě spadne nešťastně inicilizované GMM), opakujte ;-) Odevzdávejte vyexportované PDF (File -> Export as -> PDF [via LaTeX]). Dejte pozor, aby se přitom neztratil žádný obsah.

Mnoho zdaru!

1 Informace o vzniku řešení

Vyplňte následující údaje

- Jméno autora: Anton Firc
- Login autora: xfirca00
- Datum vzniku: 22.3.2021

[161]:
```
import copy
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats
from ikrlib_hw import gellipse, plot2dfun
```

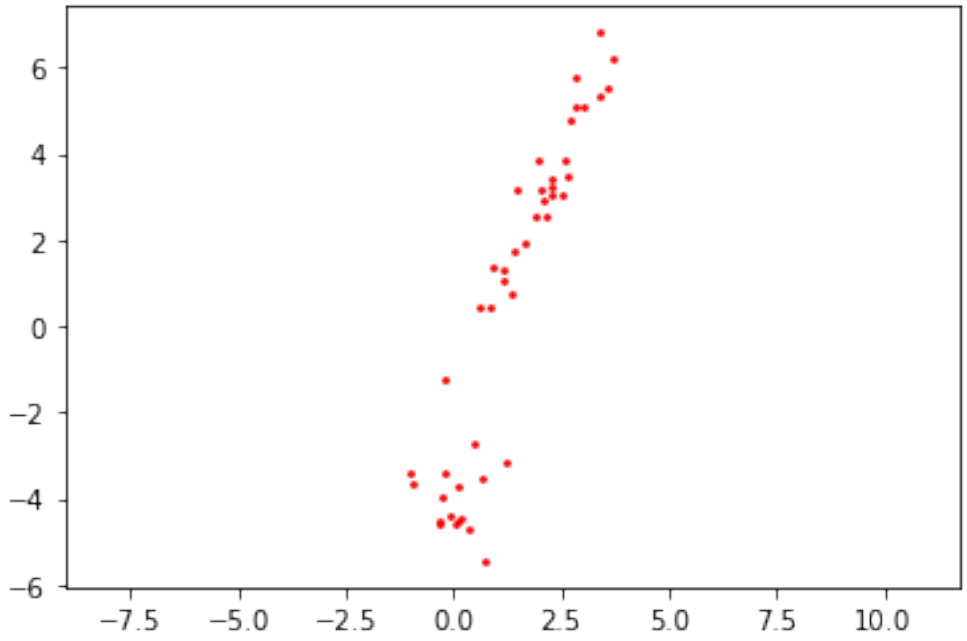
2 Modelování dat

V této kapitole vytvoříte řadu různých modelů dvoudimenzionálních dat. Základním prvkem jejich rozhraní je metoda `.pdf(data)`, která pro pole `data` o tvaru `[samples, features]` vrátí pole hodnot PDF o tvaru `[samples]`. Bude je tedy možné podrobit následující funkci pro výpočet log-likelihoodu.

[162]:
```
def log_likelihood(model, data):
    return np.sum(np.log(model.pdf(data)))
```

Následující buňka Vám načte data. V komentáři je kód, pomocí něhož byla data vytvořena, pro srovnání nebo hraní. Při odevzdání se prosím držte dodaných dat.

[163]:
```
# def provide_data():
#     mu_1 = np.asarray([2.0, 3.0])
#     cov_1 = np.asarray([[1.0, 1.9], [1.9, 4.0]])
#     samples_1 = scipy.stats.multivariate_normal(mu_1, cov_1).rvs(30)
#
#     mu_2 = np.asarray([0.0, -4.0])
#     cov_2 = np.asarray([[0.25, -0.1], [-0.1, 1.0]])
#     samples_2 = scipy.stats.multivariate_normal(mu_2, cov_2).rvs(15)
#
#     return np.concatenate([samples_1, samples_2])
#
# samples = provide_data()
# unseen_samples = provide_data()
#
# np.savetxt('modeling-train.txt', samples)
# np.savetxt('modeling-test.txt', unseen_samples)

samples = np.loadtxt('modeling-train.txt')
unseen_samples = np.loadtxt('modeling-test.txt')

plt.figure()
plt.scatter(samples[:, 0], samples[:, 1], c='r', s=3)
_ = plt.axis('equal')
```



2.1 Uniformní rozdělení

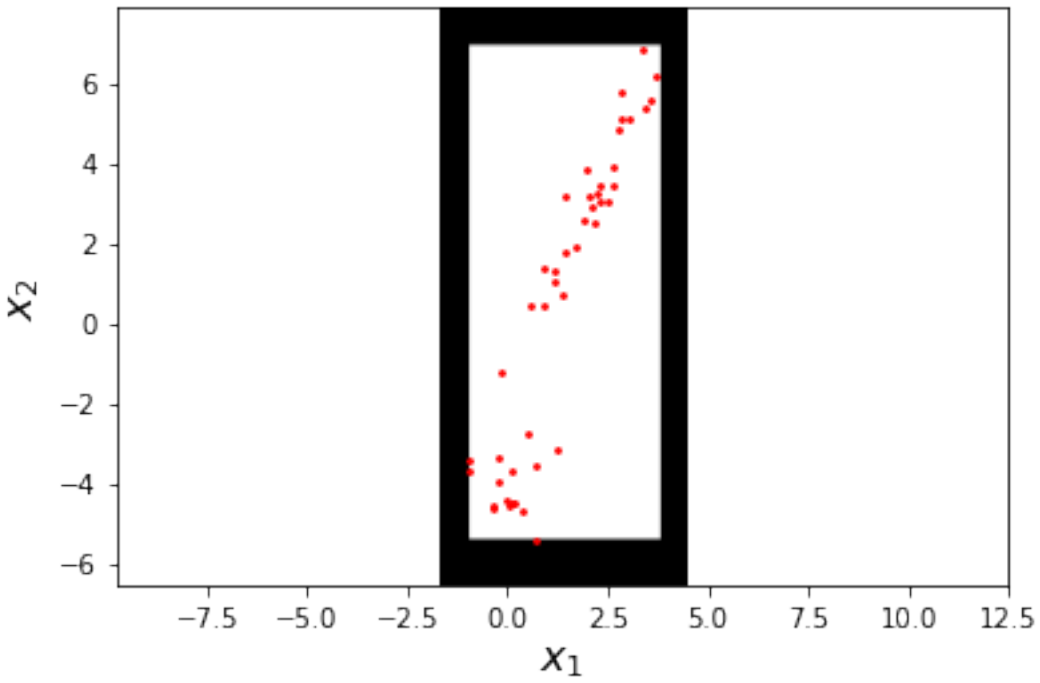
Práci začnete nejjednodušším z rozdělení: rovnoměrným. Zamysleste, jak ho parametrizovat a naimplementujte (1) metodu, která vrací dimenzionalitu modelu (`.dim()`), (2) funkci pro vytvoření maximum-likelihood odhadu a konstruktor a konečně (3) vlastní metodu pro odhad hustoty pravděpodobnosti (`.pdf()`).

1 bod

Pro kontrolu: v obrázku byste měli vidět obdélník PDF, který těsně rámuje data.

[164]:
```
class MultivariateUniform:
    def __init__(self, data):
        self.data = data
        self.data_x_min = self.data[:,0].min()
        self.data_x_max = self.data[:,0].max()
        self.data_y_min = self.data[:,1].min()
        self.data_y_max = self.data[:,1].max()
        self.x_len = self.data_x_max - self.data_x_min
        self.y_len = self.data[:,1].max() - self.data[:,1].min()
        self.value = 1 / (self.x_len * self.y_len)

    @property
    def dim(self):
        return self.data.ndim

    def pdf(self, x):
```

```
        assert x.shape[-1] == self.dim

        x_in_range = np.logical_and(self.data_x_min <= x[:,0], x[:,0] <= self.
↪data_x_max)
        y_in_range = np.logical_and(self.data_y_min <= x[:,1], x[:,1] <= self.
↪data_y_max)
        in_range = np.logical_and(x_in_range, y_in_range)
        res = np.where(in_range, self.value, 0)
        return res

    @classmethod
    def ml_estimate(cls, data):
        return cls(data)

ml_uni = MultivariateUniform.ml_estimate(samples)
plt.figure()
plt.scatter(samples[:, 0], samples[:, 1], c='r', s=3)
xmin, xmax = plt.gca().get_xlim()
ymin, ymax = plt.gca().get_ylim()
bbox = [xmin-0.5, xmax+0.5, ymin-0.5, ymax+0.5]
plot2dfun(ml_uni.pdf, bbox, resolution=100)
_ = plt.axis('equal')
```



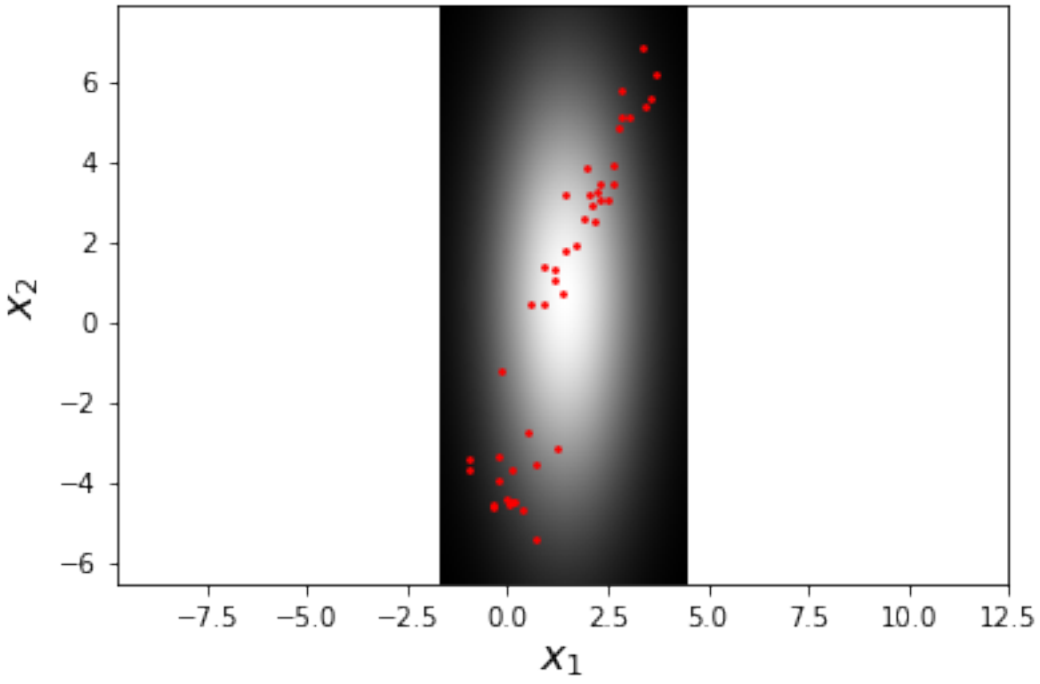
2.2 Gaussovo rozložení s diagonální kovariancí

Dalším krokem je vytvoření ortogonálního Gaussova rozložení. Je parametrizováno vektorem střední hodnoty a vektorem variancí v jednotlivých dimenzích. Naimplementujte (1) funkci pro odhad maximálně věrohodných parametrů (`.ml_estimate`, parametr `weights` přiřazuje vzorkům váhu, využijete později u GMM) a (2) hustotu rozdělení pravděpodobnosti (`.pdf()`). Z ní vyčleňte (3) výpočet jmenovatele, kterým se hodnota normalizuje (`.normalizer()`) – není funkcí dat, spíše vlastností modelu.

1 bod

[165]:
```
class DiagonalGaussian:
    def __init__(self, mean, var, data_cnt):
        shapes_str = f'mean.shape: {mean.shape}, var.shape: {var.shape}'
        assert len(mean.shape) == 1, shapes_str
        assert len(var.shape) == 1, shapes_str
        assert var.shape == mean.shape, shapes_str

        self.mean = mean
        self.var = var
        self.cov = np.array([[var[0],0.],[0.,var[1]]])
        self.data_cnt = data_cnt

    @property
    def dim(self):
        return self.mean.shape[0]

    def normalizer(self):
        return np.sqrt(((np.pi * 2) ** len(self.mean)) * np.linalg.det(self.cov))

    def pdf(self, x):
        assert x.shape[-1] == self.dim

        inv_cov = np.linalg.inv(self.cov)
        x_min_y = x - self.mean

        tst = x_min_y @ inv_cov
        rst = tst @ x_min_y.T
        dist = rst.diagonal()

        e_pow = np.exp(-0.5 * dist)

        return e_pow / self.normalizer()


    @classmethod
    def ml_estimate(cls, data, weights=None):
        if weights is not None:
            assert len(weights.shape) == 1
```

```
            assert weights.shape[0] == data.shape[0]
        else:
            weights = np.ones((data.shape[0],))

        data_cnt = data.shape[0]

        mu = np.average(data, axis=0, weights=weights)
        var = np.cov(data, rowvar=False, aweights=weights).diagonal()

        return cls(mu, var, data_cnt)

    def gellipse(self, *args, **kwargs):
        gellipse(self.mean, np.diag(self.var), *args, **kwargs)

diag_gauss = DiagonalGaussian.ml_estimate(samples)

plt.figure()
plt.scatter(samples[:, 0], samples[:, 1], c='r', s=3)
xmin, xmax = plt.gca().get_xlim()
ymin, ymax = plt.gca().get_ylim()
bbox = [xmin-0.5, xmax+0.5, ymin-0.5, ymax+0.5]
plot2dfun(diag_gauss.pdf, bbox, resolution=100)
_ = plt.axis('equal')
```



3 Plné Gaussovo rozložení

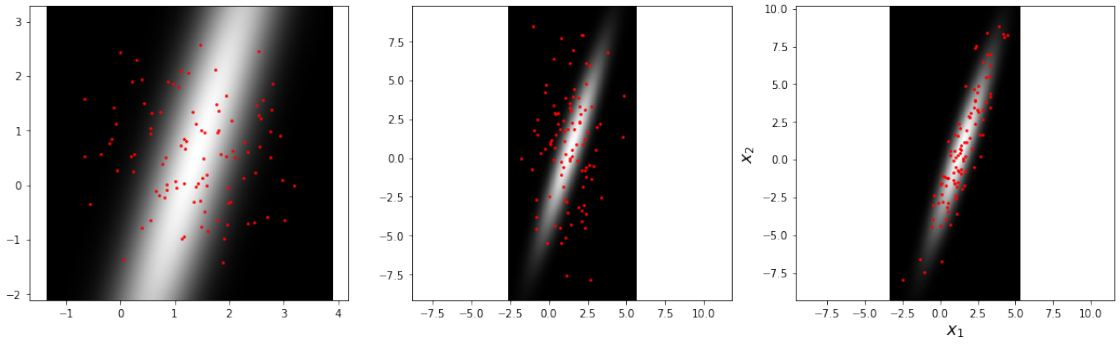
Plné Gaussovo rozložení sleduje stejnou strukturu jako ortogonální, jen Vám dá více zabrat ;-) Navíc zde naimplementujete vzorkování (`.sample()`, nepoužívejte žádné jiné vzorkování než dodané vzorkování ze standarního normálního rozložení). Kromě pochopitelného počtu vzorků, které má vytvořit, má navíc dva přepínače: `ignore_var` a `ignore_cov`. Pokud je nastaven `ignore_var`, ignorujte kovarianci úplně (předpokládejte jednotkovou matici) a vzorkuje prostě jen kolem střední hodnoty, pokud `ignore_cov`, berte v potaz pouze variance v jednotlivých dimenzích (tzn. diagonálu). Doporučujeme implementovat takto postupně.

1 bod

[166]:
```
class MultivariateGaussian:
    def __init__(self, mean, cov, data_cnt):
        shapes_str = f'mean.shape: {mean.shape}, cov.shape: {cov.shape}'
        assert len(mean.shape) <= 2, shapes_str
        assert len(cov.shape) == 2, shapes_str
        assert cov.shape[0] == cov.shape[1], shapes_str
        assert mean.squeeze().shape[0] == cov.shape[0], shapes_str

        self.mean = mean.squeeze()
        self.cov = cov
        self.data_cnt = data_cnt

    @property
    def dim(self):
        return self.cov.shape[0]

    def pdf(self, x):
        assert x.shape[-1] == self.dim

        inv_cov = np.linalg.inv(self.cov)

        x_min_y = x - self.mean
        lhs = x_min_y @ inv_cov
        res = lhs @ x_min_y.T
        dist = res.diagonal()

        e_pow = np.exp(-0.5 * dist)

        return e_pow / self.normalizer()

    def normalizer(self):
        return np.sqrt(((np.pi * 2) ** len(self.mean)) * np.linalg.det(self.cov))

    @classmethod
    def ml_estimate(cls, data, weights=None):
        if weights is not None:
```

```
            assert len(weights.shape) == 1
            assert weights.shape[0] == data.shape[0]
        else:
            weights = np.ones((data.shape[0],))

        data_cnt = data.shape[0]

        mu = np.average(data, axis=0, weights=weights)
        cov = np.cov(data, rowvar=False, aweights=weights)
        return cls(mu, cov, data_cnt)


    def gellipse(self, *args, **kwargs):
        gellipse(self.mean, self.cov, *args, **kwargs)

    def sample(self, nb_samples, ignore_var=False, ignore_cov=False):
        std_norm_noise = scipy.stats.norm.rvs(size=(nb_samples, self.dim))

        if ignore_var:
            std_norm_mean = np.mean(std_norm_noise, axis=0)
            correction = std_norm_mean - self.mean
            return std_norm_noise - correction
        elif ignore_cov:
            cov = np.array([[self.cov[0][0], 0.],[0., self.cov[1][1]]])
            d,v = np.linalg.eigh(cov)
            return (std_norm_noise * np.sqrt(d)).dot(v) + self.mean
        else:
            d,v = np.linalg.eigh(self.cov)
            return (std_norm_noise * np.sqrt(d)).dot(v) + self.mean


ml_gauss = MultivariateGaussian.ml_estimate(samples)

samples_mean_only = ml_gauss.sample(100, ignore_var=True)
samples_no_cov = ml_gauss.sample(100, ignore_cov=True)
samples_full = ml_gauss.sample(100)

def plot_pdf_with_samples(ax, pdf_function, samples):
    ax.scatter(samples[:, 0], samples[:, 1], c='r', s=3)
    xmin, xmax = ax.get_xlim()
    ymin, ymax = ax.get_ylim()
    bbox = [xmin-0.5, xmax+0.5, ymin-0.5, ymax+0.5]
    plot2dfun(pdf_function, bbox, resolution=100, ax=ax)
    ax.axis('equal')

fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 5))
plot_pdf_with_samples(ax1, ml_gauss.pdf, samples_mean_only)
```

```
plot_pdf_with_samples(ax2, ml_gauss.pdf, samples_no_cov)
plot_pdf_with_samples(ax3, ml_gauss.pdf, samples_full)

plt.show()
```



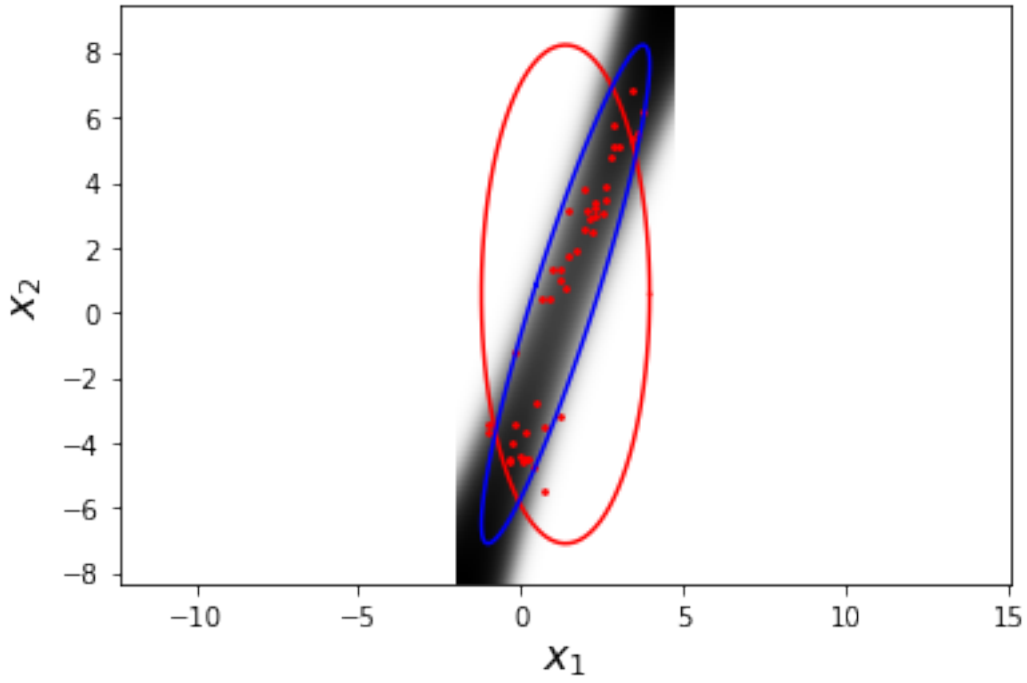
3.1 Směsi gaussovek

Nyní se postupně dopracujete k modelování pomocí směsi Gaussových rozložení. Prvním krokem je určování příslušnosti dat k modelům, což je zodpovědnost funkce `posteriors`. Dostává `data` (`[samples, features]`), `models` (seznam objektů) a `priors` (`[models]`). Vrací posteriorní pravděpodobnosti, že dato vzešlo z modelu, uspořádané jako `[models, samples]`.

1 bod

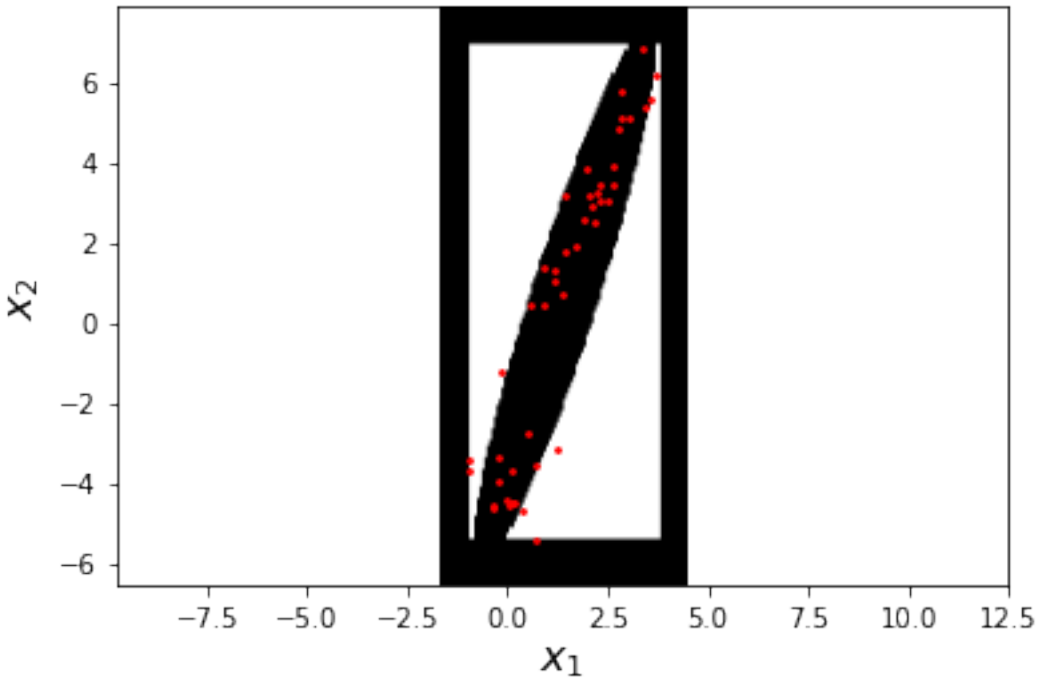
[167]:
```
def posteriors(data, models, priors):
    result = []

    for i in range(0, len(models)):
        result.append(models[i].pdf(data) * priors[i])

    pdfs = np.stack(result)
    pdf_sums = pdfs.sum(axis=0)

    return pdfs / pdf_sums

priors = np.asarray([0.5, 0.5])
data = np.asarray([[3.0, 1.0], [2.0, 3.0], [0.0, 0.0]])

plt.figure()
plt.scatter(samples[:, 0], samples[:, 1], c='r', s=3)
diag_gauss.gellipse(color='r')
ml_gauss.gellipse(color='b')

xmin, xmax = plt.gca().get_xlim()
ymin, ymax = plt.gca().get_ylim()
```

```
bbox = [xmin-0.5, xmax+0.5, ymin-0.5, ymax+0.5]
plot2dfun(
    lambda data: posteriors(data, [diag_gauss, ml_gauss], np.asarray([0.5, 0.
 ↪5]))[0],
    bbox,
    resolution=100
)
_ = plt.axis('equal')
```



Pokud chcete tvrdé rozhodnutí, pro dva modely stačí prahovat:

[168]:
```
plt.figure()
plt.scatter(samples[:, 0], samples[:, 1], c='r', s=3)
xmin, xmax = plt.gca().get_xlim()
ymin, ymax = plt.gca().get_ylim()
bbox = [xmin-0.5, xmax+0.5, ymin-0.5, ymax+0.5]
plot2dfun(
    lambda data: (posteriors(data, [ml_uni, ml_gauss], priors)[0] >= 0.5).
 ↪astype(float),
    bbox,
    resolution=100
)
_ = plt.axis('equal')
```



3.2 Reprezentace GMM

je celkem malá třída, která v podstatě jen drží seznam gaussovek a pole jejich vah. Naimplementujte do ní jako obvykle (1) metodu `.pdf()` Zajímavější je pak (2) funkce `initialize`, která na dodaných `data` vytvoří `nb_compomenents` modelů třídy `compomenent_cls`. Je na vás, jak to uděláte, jen by se tu nemělo odehrávat žádné trénovaní. Nejspíše si ve funkci náhodně vyberete (`np.random.choice`) několik málo vzorků pro každou komponentu a na nich odhadnete její parametry; váhy inicializujte rovnoměrně.

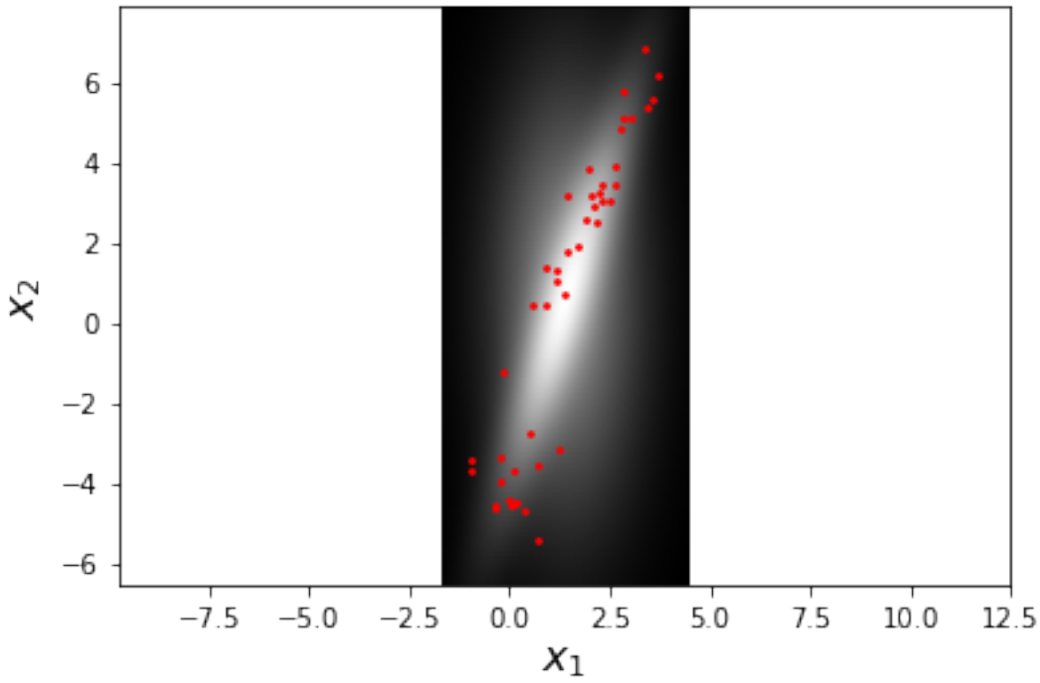
1 bod

[169]:
```
class GMM:
    def __init__(self, gaussians, weights):
        assert len(gaussians) == len(weights)
        assert abs(np.sum(weights) - 1.0) < 1e-4

        self.gaussians = gaussians
        self.weights = weights

    def pdf(self, data):
        assert len(self.gaussians) == len(self.weights)

        results = []

        for i in range(0,len(self.gaussians)):
```

```
            results.append(self.gaussians[i].pdf(data) * self.weights[i])

        results_np = np.stack(results)

        return results_np.sum(axis=0)

    def gellipses(self, *args, **kwargs):
        for m in self.gaussians:
            m.gellipse(*args, **kwargs)

    @classmethod
    def initialize(cls, data, compoment_cls, nb_components):
        components = []

        for i in range(0, nb_components):
            samples = np.array([np.random.choice(data[:,0], 3), np.random.
↪choice(data[:,1], 3)]).T
            components.append(compoment_cls.ml_estimate(samples))

        priors = np.ones(nb_components) * 1/nb_components
        return cls(components, priors)
```

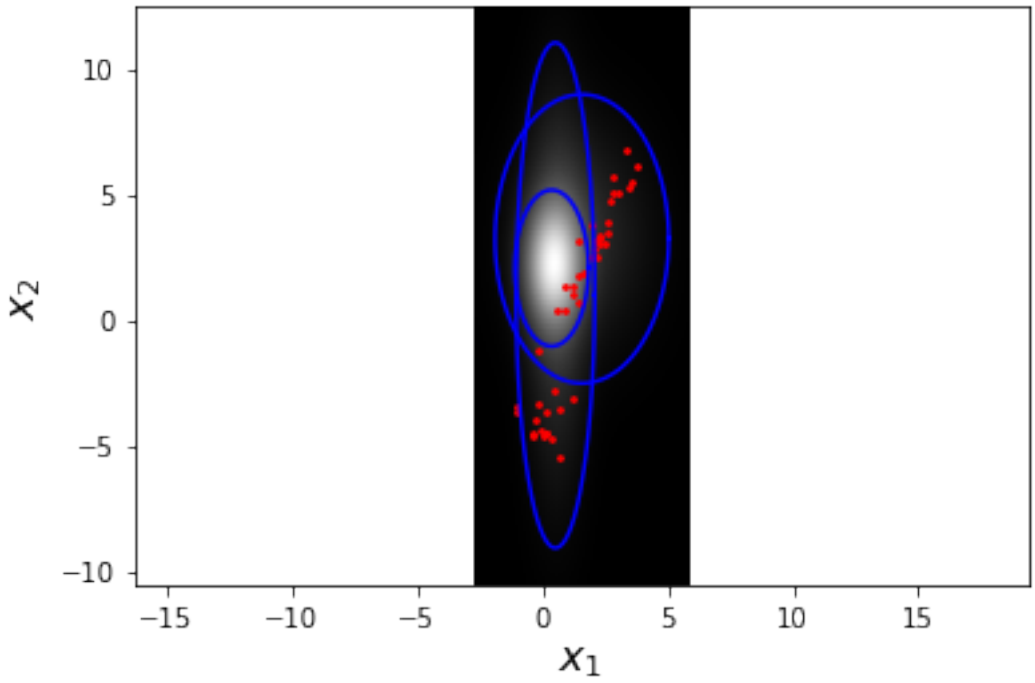
Zde si můžete s ručně vytvořeným GMM ověřit, že máte v pořádku `GMM.pdf()`.

[170]:
```
gmm = GMM([diag_gauss, ml_gauss], np.asarray([0.75, 0.25]))

plt.figure()
plt.scatter(samples[:, 0], samples[:, 1], c='r', s=3)
# diag_gauss.gellipse(color='r')
# ml_gauss.gellipse(color='b')

xmin, xmax = plt.gca().get_xlim()
ymin, ymax = plt.gca().get_ylim()
bbox = [xmin-0.5, xmax+0.5, ymin-0.5, ymax+0.5]
plot2dfun(
    gmm.pdf,
    bbox,
    resolution=100
)
_ = plt.axis('equal')
```



A tady se můžete mrknout, co dělá `GMM.initialize()`.

[171]:
```
gmm_init = GMM.initialize(samples, DiagonalGaussian, 3)

plt.figure()
plt.scatter(samples[:, 0], samples[:, 1], c='r', s=3)
gmm_init.gellipses(c='b')
xmin, xmax = plt.gca().get_xlim()
ymin, ymax = plt.gca().get_ylim()
bbox = [xmin-0.5, xmax+0.5, ymin-0.5, ymax+0.5]
plot2dfun(
    gmm_init.pdf,
    bbox,
    resolution=100
)
_ = plt.axis('equal')
```



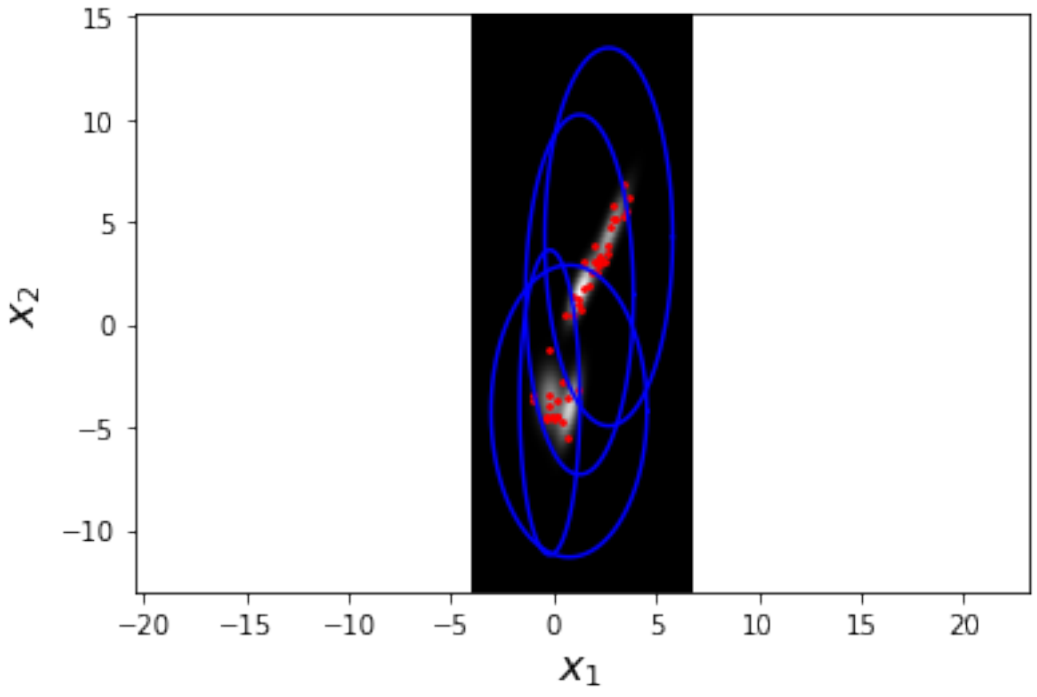
3.3 Viterbiho trénování

Naimplementujte funkci `viterbi_train()`, která provede `nb_iters` iterací Viterbiho trénování modelu `gmm` na dodaných `data`. Nebojte se sahat dovnitř `gmm`, normálně by tohle byla členská metoda. Tady je zvlášť, abychom Vás ušetřili skrolování mezi implementací a použitím. Je možné, že někdy některá komponenta "prohraje", nebude schopná se updatovat z dat, která jí byla přiřazena. V tom případě trénovaní přerušte a vraťte poslední validní model.

1 bod

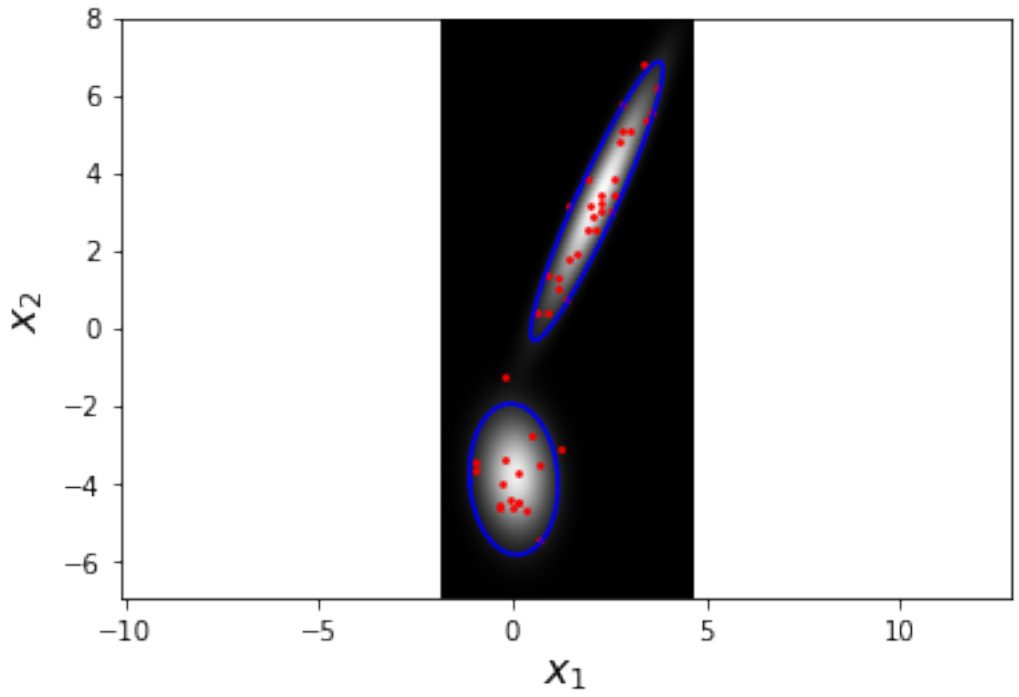
[217]:
```
def viterbi_train(gmm, data, nb_iters):
    for f in range(nb_iters):
        gmm_backup = gmm
        post = posteriors(data, gmm.gaussians, gmm.weights)
        idxs = np.argmax(post, axis=0)

        lab_data = np.array([data[:,0], data[:,1], idxs]).T

        for i in range(0, len(gmm.gaussians)):
            gauss_idx = np.where(lab_data[:,2] == i)[0]
            gauss_data = np.take(data, gauss_idx, axis=0)

            if len(gauss_data) == 0:
                return gmm_backup
```

```
            gmm.gaussians[i].cov = np.cov(gauss_data, rowvar=False)
            gmm.gaussians[i].mean = np.array([np.mean(gauss_data[:,0]),np.
 ↪mean(gauss_data[:,1])])

    return gmm

init = GMM.initialize(samples, DiagonalGaussian, 4)
gmm_diag_viterbi = viterbi_train(init, samples, nb_iters=10)

plt.figure()
plt.scatter(samples[:, 0], samples[:, 1], c='r', s=3)
gmm_diag_viterbi.gellipses(c='b')
xmin, xmax = plt.gca().get_xlim()
ymin, ymax = plt.gca().get_ylim()
bbox = [xmin-0.5, xmax+0.5, ymin-0.5, ymax+0.5]
plot2dfun(
    gmm_diag_viterbi.pdf,
    bbox,
    resolution=100
)
_ = plt.axis('equal')
plt.show()
```



Následující buňka provede Viterbiho trénování dvou Gaussovek s plnou kovariancí.

[173]:
```
init = GMM.initialize(samples, MultivariateGaussian, 2)
gmm_full_viterbi = viterbi_train(init, samples, nb_iters=10)

plt.figure()
plt.scatter(samples[:, 0], samples[:, 1], c='r', s=3)
gmm_full_viterbi.gellipses(c='b')
xmin, xmax = plt.gca().get_xlim()
ymin, ymax = plt.gca().get_ylim()
bbox = [xmin-0.5, xmax+0.5, ymin-0.5, ymax+0.5]
plot2dfun(
    gmm_full_viterbi.pdf,
    bbox,
    resolution=100
)
_ = plt.axis('equal')
plt.show()
```



3.4 Expectation-Maximization trénování

Konečně se dostáváte k zlatému hřebu kapitoly – EM trénování GMMka. E-step vám de-facto zařizuje funkce `posteriors`. Nyní naimplementujte M-step, který aktualizuje jednotlivé komponenty GMM (klidně je nahraďte novými ML odhady, zde konečně využijete parametr `weights`).

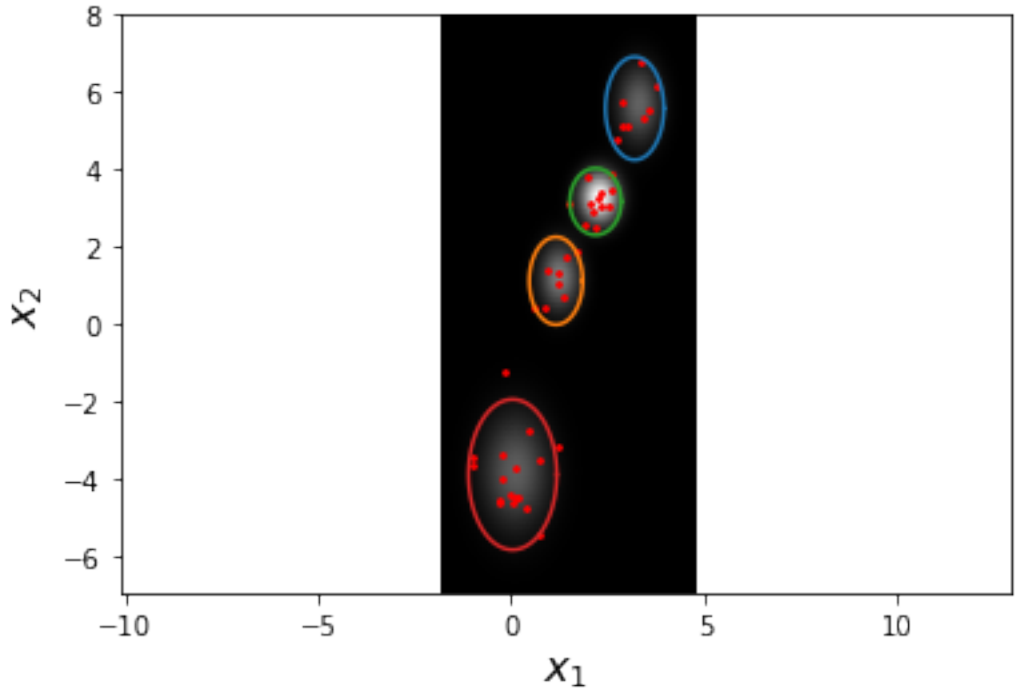
V samotné funkci `em_train` se stačí volat E-step a M-step a starat se o bookkeeping průběžných

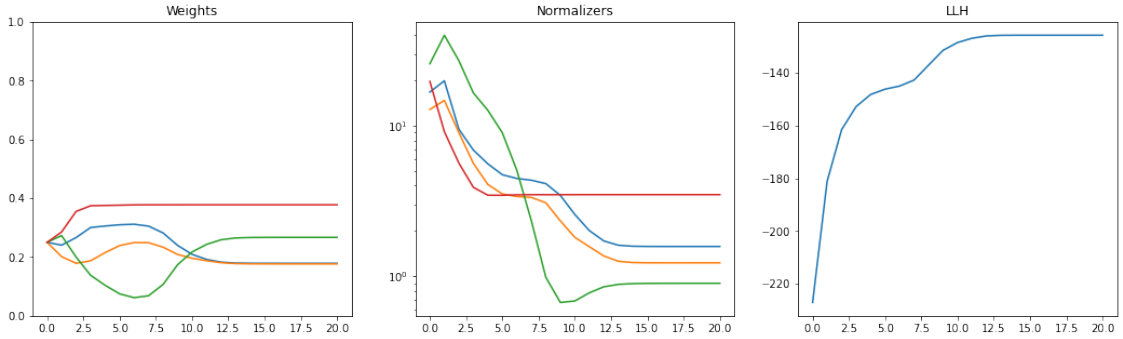
hodnot: Log-likelihood dat s aktuálním modelem a váhy a normalizery jednotlivých komponent (vracejte jako `[komponenta, hodnoty]`) – budou se kreslit.

2 body

[185]:
```
def m_step(gmm, data, responsibilities):
    for i in range(len(gmm.gaussians)):
        gmm.gaussians[i] = gmm.gaussians[i].ml_estimate(data, responsibilities[:
 ↪,i])
        gmm.weights[i] = responsibilities[:,i].sum() / data.shape[0]

def em_train(gmm, data, nb_iters):
    llhs = [log_likelihood(gmm, data)]
    weights = [gmm.weights.copy()]
    normalizers = [np.asarray([g.normalizer() for g in gmm.gaussians])]

    for _ in range(nb_iters):
        e_step = posteriors(data, gmm.gaussians, gmm.weights).T
        m_step(gmm, data, e_step)

        llhs.append(log_likelihood(gmm, data))
        weights.append(gmm.weights.copy())
        normalizers.append(np.asarray([g.normalizer() for g in gmm.gaussians]))

    return gmm, llhs, np.stack(weights).T, np.stack(normalizers).T

def plot_gmm_training(gmm, llhs, weights, normalizers):
    plt.figure()
    plt.scatter(samples[:, 0], samples[:, 1], c='r', s=3)
    gmm.gellipses()
    xmin, xmax = plt.gca().get_xlim()
    ymin, ymax = plt.gca().get_ylim()
    bbox = [xmin-0.5, xmax+0.5, ymin-0.5, ymax+0.5]
    plot2dfun(
        gmm.pdf,
        bbox,
        resolution=100
    )
    _ = plt.axis('equal')

    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 5))

    for w in weights:
        ax1.plot(w)
    ax1.set_ylim([0.0, 1.0])
    ax1.set_title('Weights')

    for n in normalizers:
```

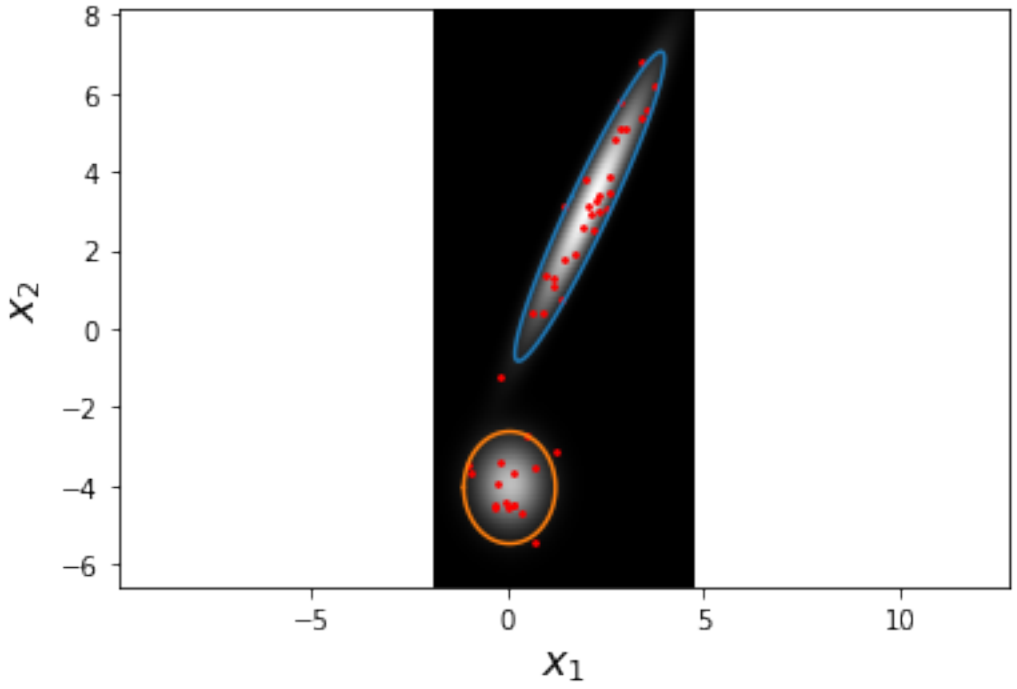
```
        ax2.plot(n)
    ax2.set_yscale('log')
    ax2.set_title('Normalizers')

    ax3.plot(llhs)
    ax3.set_title('LLH')

    plt.show()

init = GMM.initialize(samples, DiagonalGaussian, 4)
gmm_diag_em, llhs, weights, normalizers = em_train(init, samples, nb_iters=20)
plot_gmm_training(gmm_diag_em, llhs, weights, normalizers)
```

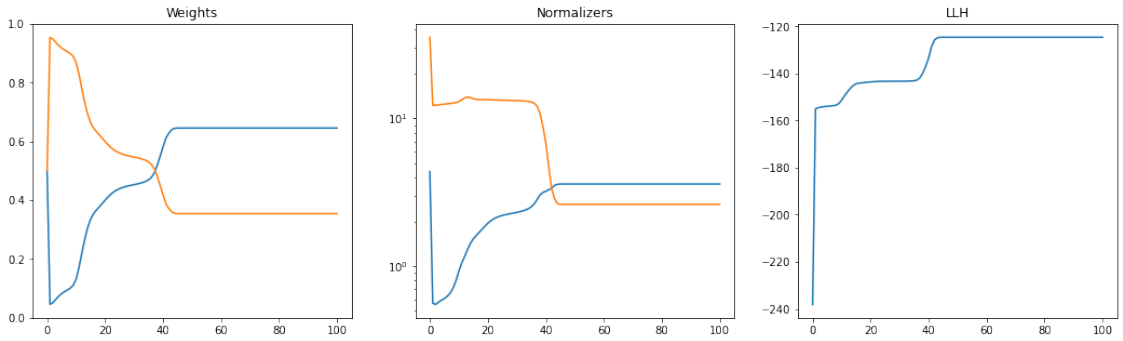




Následující buňka jen natrénuje směs gaussovek s plnou kovariancí.

[188]:
```
init = GMM.initialize(samples, MultivariateGaussian, 2)
gmm_full_em, llhs, weights, normalizers = em_train(init, samples, nb_iters=100)
plot_gmm_training(gmm_full_em, llhs, weights, normalizers)
```





Konečně srovnejte všechny natrénované modely na viděných i neviděných datech.

[218]:
```
models = [
    ('uniform', ml_uni),
    ('gauss_diag', diag_gauss),
```

```
    ('gauss_full', ml_gauss),
    ('gmm_init', gmm_init),
    ('gmm_diag_vite', gmm_diag_viterbi),
    ('gmm_full_vite', gmm_full_viterbi),
    ('gmm_diag_em', gmm_diag_em),
    ('gmm_full_em', gmm_full_em),
]

print('model\t\tSeen\tUnseen')
print('-'*30)
for name, model in models:
    train_llh = log_likelihood(model, samples)/len(samples)
    test_llh = log_likelihood(model, unseen_samples)/len(unseen_samples)
    print(f'{name}:\t{train_llh:.2f}\t{test_llh:.2f}')
```

```
model           Seen    Unseen
------------------------------
uniform:        -4.06   -inf
gauss_diag:     -4.42   -4.39
gauss_full:     -3.47   -3.62
gmm_init:       -4.64   -4.55
gmm_diag_vite:  -2.93   -3.20
gmm_full_vite:  -2.84   -3.09
gmm_diag_em:    -2.80   -3.49
gmm_full_em:    -2.77   -3.08
```

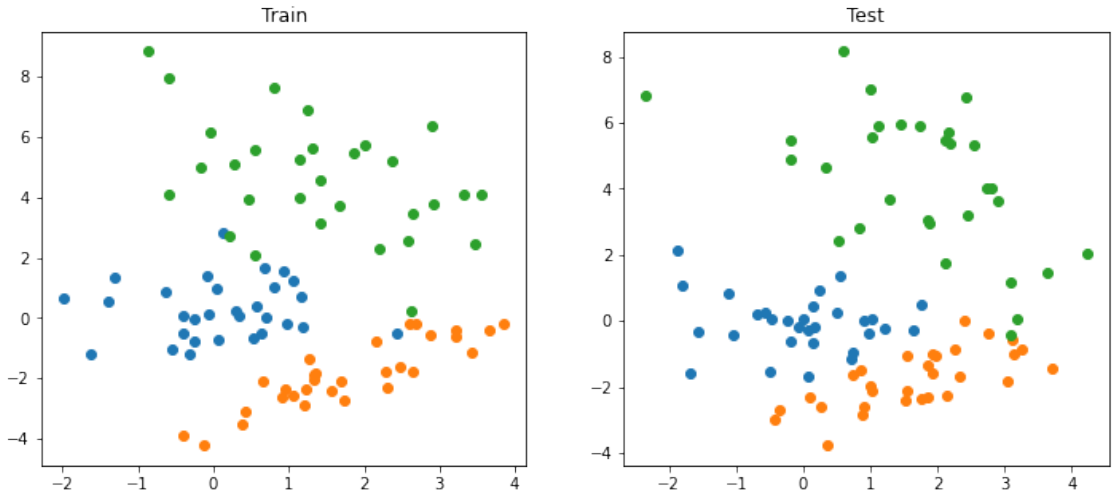
```
<ipython-input-162-1ae664309d48>:2: RuntimeWarning: divide by zero encountered
in log
  return np.sum(np.log(model.pdf(data)))
```

4 Jednoduché klasifikátory

V této části vytvoříte několik jednoduchých klasifikátorů na dodaných datech. Opět prosím odevzdávejte řešení na dodaných datech.

[177]:
```
# sources = [
#     (np.array([0,    0]),   np.array([[1.0, 0.0], [0.0, 1.0]])),
#     (np.array([1.5, -2.0]), np.array([[1.0, 0.8], [0.8, 1.0]])),
#     (np.array([1.5,  4.5]), np.array([[1.5, -1.2], [-1.2, 3.5]])),
# ]
#
# generators = [scipy.stats.multivariate_normal(mu, cov) for mu, cov in sources]
# train_samples = [g.rvs(30) for g in generators]
# test_samples = [g.rvs(30) for g in generators]
#
# for i, cls in enumerate(train_samples):
#     np.savetxt(f'classification-train-cls_{i}.txt', cls)
```

```
#
# for i, cls in enumerate(test_samples):
#     np.savetxt(f'classification-test-cls_{i}.txt', cls)

def targeted_from_per_class(per_class_samples):
    X = np.concatenate(test_samples)
    t = np.concatenate([i*np.ones(len(cls)) for (i, cls) in␣
 ↪enumerate(test_samples)])
    return X, t

train_samples = [np.loadtxt(f'classification-train-cls_{i}.txt') for i in␣
 ↪range(3)]
test_samples = [np.loadtxt(f'classification-test-cls_{i}.txt') for i in range(3)]

train_x, train_t = targeted_from_per_class(train_samples)
complete_test_x, complete_test_t = targeted_from_per_class(test_samples)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.set_title('Train')
for cls in train_samples:
    ax1.scatter(cls[:,0], cls[:,1])

ax2.set_title('Test')
for cls in test_samples:
    ax2.scatter(cls[:,0], cls[:,1])
plt.show()
```



4.0.1 Rozhraní klasifikátoru

Každý klasifikátor musí být volatelný (`__call__()`) a vracet index vítězné třídy pro každé dato. Kvantitativní vyhodnocení bude probíhat pomocí následující funkce `evaluate()`. Druhá očekávaná metoda je `.posteriors()`, která vrátí posteriorní pravděpodobnosti jednotlivých tříd, ve stejném formátu jako výše zavedená funkce `posteriors()`.

[178]:
```
def evaluate(classifier, data, targets):
    preds = classifier(data)
    hits = preds == targets
    return np.sum(hits) / np.prod(data.shape[:-1])
```

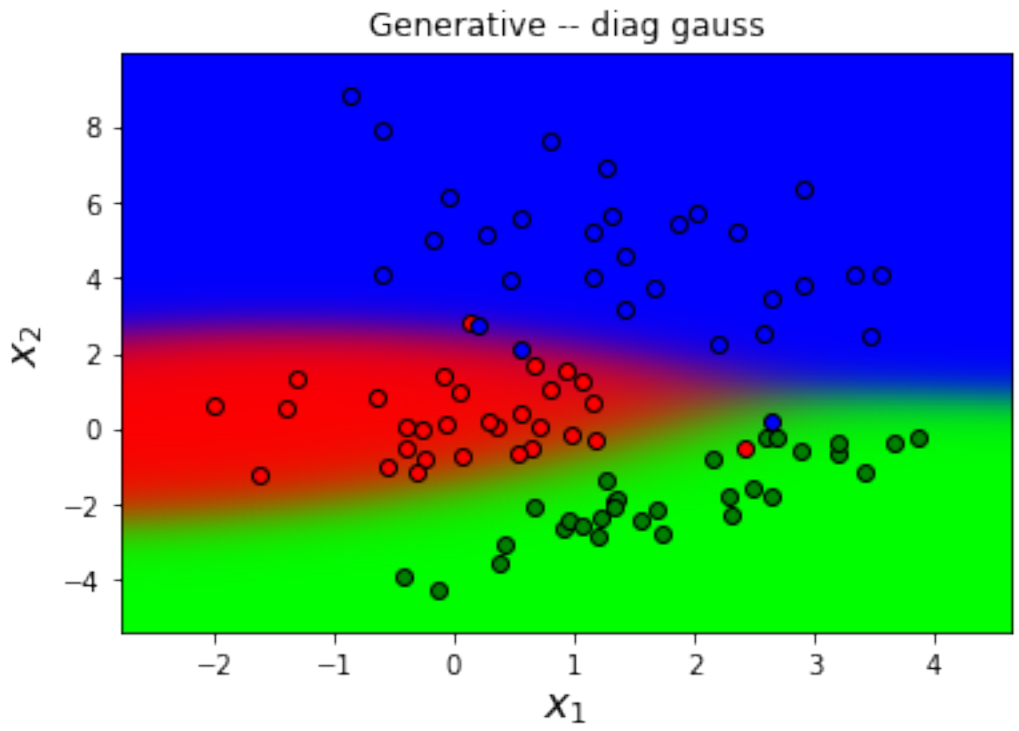
4.1 Generativní klasifikátor

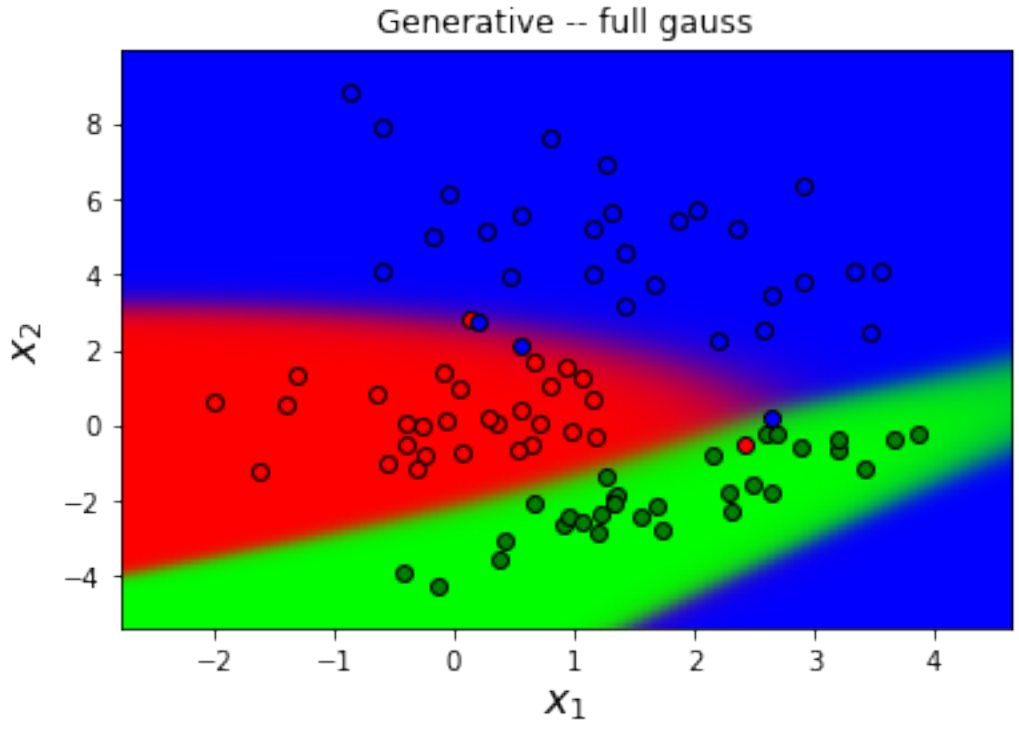
Generativní klasifikátor je v podstatě jen kontejner obsahující modely tříd a jejich priory. Naimplementujte vlastní klasifikaci v `.__call__()` (využijete `np.argmax()`).

1 bod

[179]:
```
class GenerativeClassifier:
    def __init__(self, models, priors):
        assert len(priors.shape) == 1
        assert len(models) == priors.shape[0]
        assert abs(np.sum(priors) - 1.0) < 1e-5

        self.models = models
        self.priors = priors

    def posteriors(self, data):
        return posteriors(data, self.models, self.priors)

    def __call__(self, data):
        y_k_arr = []

        for i in range(len(self.models)):
            w_k = np.linalg.inv(self.models[i].cov) @ self.models[i].mean
            w_0 = (-0.5 * self.models[i].mean @ np.linalg.inv(self.models[i].
 ↪cov) @ self.models[i].mean.T) + np.log(self.priors[i])

            y_k_arr.append(w_k @ data.T + w_0)

        y_k = np.stack(y_k_arr, axis=1)
        return y_k.argmax(axis=1)

priors = np.ones((len(train_samples),)) / len(train_samples)

classifier_ml_diag_gaussians = GenerativeClassifier(
    [DiagonalGaussian.ml_estimate(cls) for cls in train_samples],
```

```
    priors,
)

classifier_ml_full_gaussians = GenerativeClassifier(
    [MultivariateGaussian.ml_estimate(cls) for cls in train_samples],
    priors,
)

def plot_classifier(name, classifier, data):
    plt.figure()
    for cls, color in zip(data, 'rgb'):
        plt.scatter(cls[:,0], cls[:,1], c=color, edgecolors='black')

    xmin, xmax = plt.gca().get_xlim()
    ymin, ymax = plt.gca().get_ylim()
    bbox = [xmin-0.5, xmax+0.5, ymin-0.5, ymax+0.5]
    plot2dfun(
        lambda X: classifier.posteriors(X).T,
        bbox,
        resolution=100
    )
    plt.title(name)
    plt.show()

plot_classifier('Generative -- diag gauss', classifier_ml_diag_gaussians,␣
 ↪train_samples)
plot_classifier('Generative -- full gauss', classifier_ml_full_gaussians,␣
 ↪train_samples)
```


Generative -- diag gauss
8
6
4
2
0
−2
−4
−2
−1
0
1
2
3
4
x_2
x_1

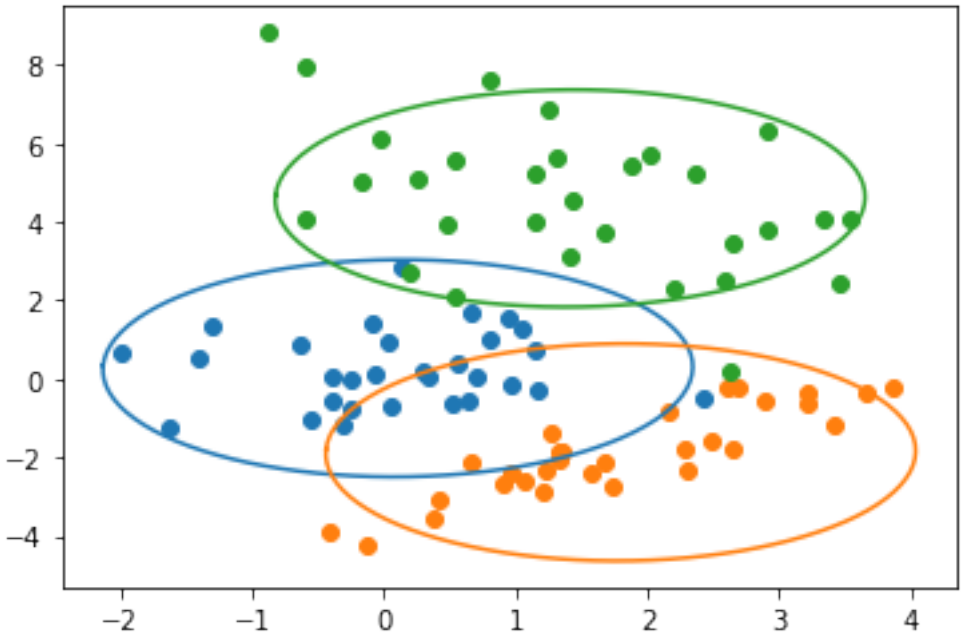

Generative -- full gauss
8
6
4
2
0
−2
−4
−2
−1
0
1
2
3
4
x_2
x_1

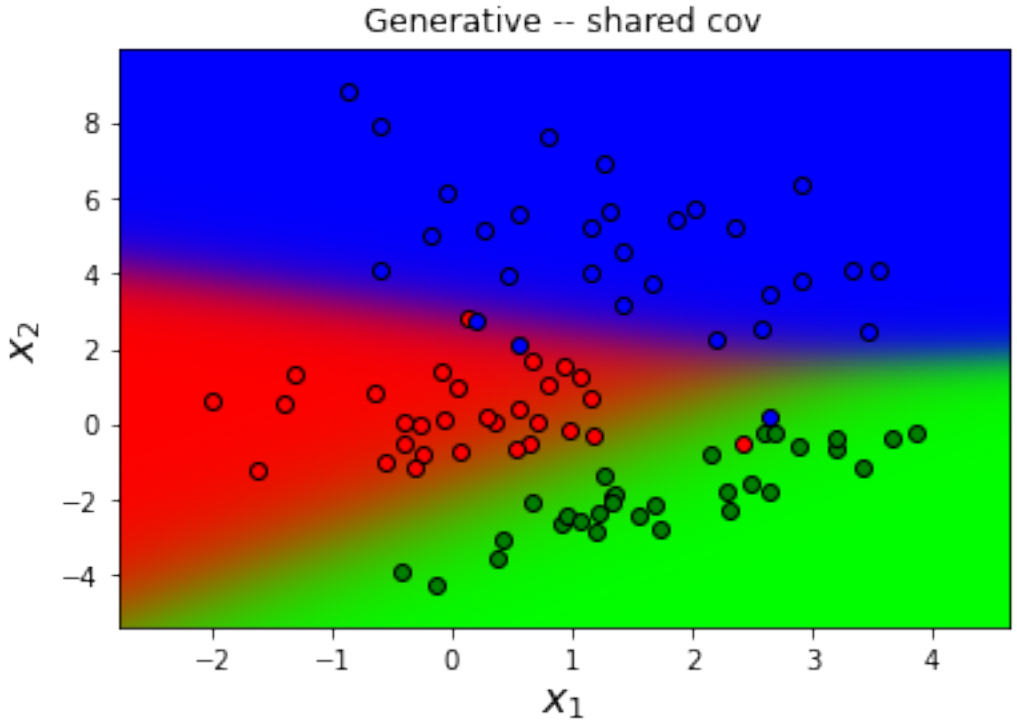
4.2 Lineární gaussovský klasifikátor

Naimplementujte funkci `estimate_glc`, která vrátí seznam gaussovek s plnou kovariancí. Střední hodnoty budou odpovídat jednotlivým třídám, ale kovarianci budou sdílet (tak, aby platilo `g1.cov is g2.cov`).

1 bod

[180]:
```
def estimate_glc(samples_by_class):
    gaussians = []
    shared_cov = np.zeros((2,2))

    for sample_class in samples_by_class:
        class_gaussian = MultivariateGaussian.ml_estimate(sample_class)
        shared_cov += class_gaussian.cov
        gaussians.append(class_gaussian)

    shared_cov = shared_cov / len(samples_by_class)

    for gaussian in gaussians:
        gaussian.cov = shared_cov

    return gaussians

shared_cov_gaussians = estimate_glc(train_samples)

plt.figure()

for g in shared_cov_gaussians:
    g.gellipse()
for cls in train_samples:
    plt.scatter(cls[:,0], cls[:,1])
plt.show()

linear_classifier = GenerativeClassifier(shared_cov_gaussians, priors)
plot_classifier('Generative -- shared cov', linear_classifier, train_samples)
```


8
6
4
2
0
−2
−4
−2
−1
0
1
2
3
4


Generative -- shared cov
8
6
4
2
0
−2
−4
x_2
−2
−1
0
1
2
3
4
x_1

4.3 Vícetřídní logistická regrese

Konečně se dostáváte k diskriminativnímu modelu. Ve třídě `MulticlassLogisticRegression` naimplementujte: `.logits()`, která vypočítá nenormalizované log-pravděpodobnosti tříd, `.posteriors`, která z nich spočítá pravděpodobnosti pomocí dodané funkce `softmax()` a klasicky zase `.__call__()`. S těmito metodami můžete pokračovat k dalšímu cvičení.

1 bod

Později bude potřeba i metoda `.gd_step()`, která provede jeden krok gradientního sestupu na daných datech `X` s třídami `t`. Velikost kroku je řízena pomocí learning rate `lr`. Bude se vám hodit transformace targetů na 1-hot kódování (`one_hot()`). Metoda upraví váhy a biasy a vrátí negative log-likelihood loss (cross-entropii), normalizovanou počtem vzorků. Povšimněte si, že i když o ní mluvíme o gradientním sestupu, dala by se implementace použít i pro Stochastic GD – prostě by ji stačilo zavolat s jediným vzorkem (nebo minibatchí).

2 body

[181]:
```
def softmax(logits, axis=-1):
    logits -= np.max(logits, axis=axis, keepdims=True)
    unnorm_probs = np.exp(logits)

    return unnorm_probs / np.sum(unnorm_probs, axis=axis, keepdims=True)

def one_hot(indexes, nb_classes=None):
    indexes = indexes.astype(int)
    if nb_classes is None:
        nb_classes = max(indexes)+1

    one_hot = np.zeros((len(indexes), nb_classes), dtype=indexes.dtype)
    one_hot[np.arange(len(indexes)), indexes] = 1
    return one_hot

class MulticlassLogisticRegression:
    def __init__(self, weights, biases):
        assert len(weights.shape) == 2
        assert len(biases.shape) == 1
        assert weights.shape[0] == biases.shape[0]
        self.w = weights
        self.b = biases[:, np.newaxis]

    def logits(self, X):
        log_probs = []

        for weight in self.w:
            log_probs.append(weight @ X.T)

        return np.stack(log_probs, axis=1) + self.b.T
```

```
    def posteriors(self, X):
        return softmax(self.logits(X)).T

    def __call__(self, X):
        probs = self.posteriors(X)
        return probs.argmax(axis=0)

    def gd_step(self, X, t, lr):
        preds = self.posteriors(X)
        hots = one_hot(t)
        ce = - hots.dot(np.log(preds)).diagonal()

        x_grad = (- (1 - preds) * X[:,0]) * hots.T
        x_grad_sum = x_grad.sum(axis=1) * lr
        self.w[:,0] = self.w[:,0] - x_grad_sum

        y_grad = - (1 - preds) * X[:,1] * hots.T
        y_grad_sum = y_grad.sum(axis=1) * lr
        self.w[:,1] = self.w[:,1] - y_grad_sum

        b_grad = - (1 - preds) * self.b * hots.T
        b_grad_sum = (b_grad.sum(axis=1) * lr).reshape(self.b.shape)
        self.b = self.b - b_grad_sum

        return ce.sum() / len(X)
```

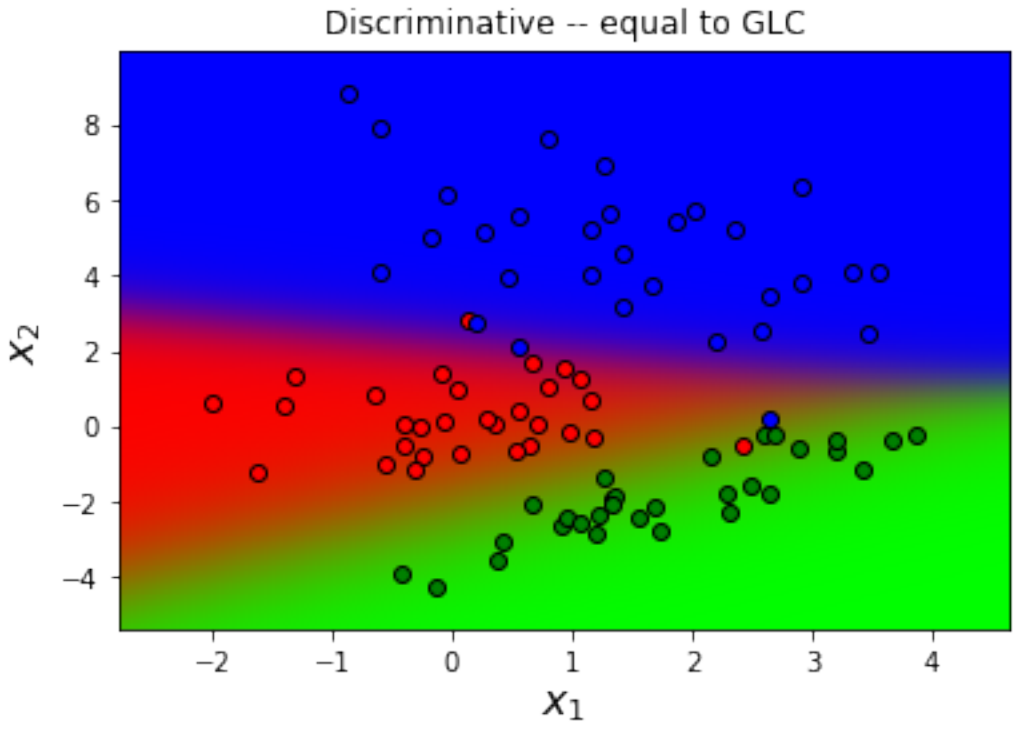
4.3.1 Transformace lineárního generativního modelu na diskriminativní

Naimplementujte funkci `softmax_regr_from_tied_gaussians`, která transformuje dodaný seznam gaussovek (se sdílenou kovariancí) na instanci `MulticlassLogisticRegression`. Výstup klasifikátoru musí být identický jako výše.

1 bod

[208]:
```
def softmax_regr_from_tied_gaussians(gaussians):
    weight_arr = []
    bias_arr = []
    centre = np.array([0., 0.])

    for g in gaussians:
        assert g.cov is gaussians[0].cov
        weight_arr.append(g.mean / 2)
        bias_arr.append( - np.linalg.norm(centre - g.mean))

    projs = np.stack(weight_arr)
    biases = np.stack(bias_arr)

    return MulticlassLogisticRegression(projs, biases)
```

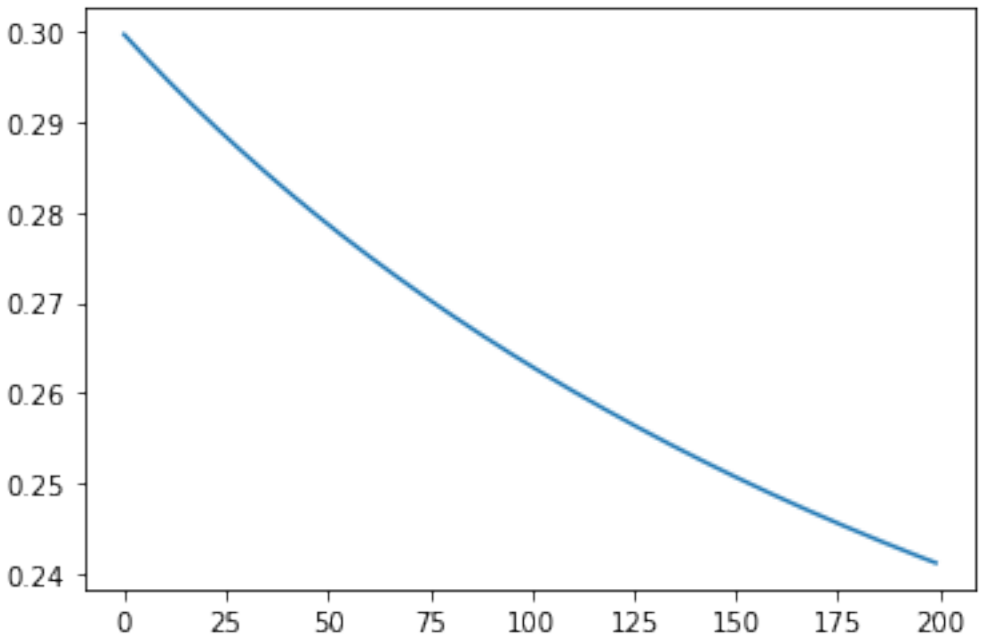
```
init_disc = softmax_regr_from_tied_gaussians(shared_cov_gaussians)
plot_classifier('Discriminative -- equal to GLC', init_disc, train_samples)
```

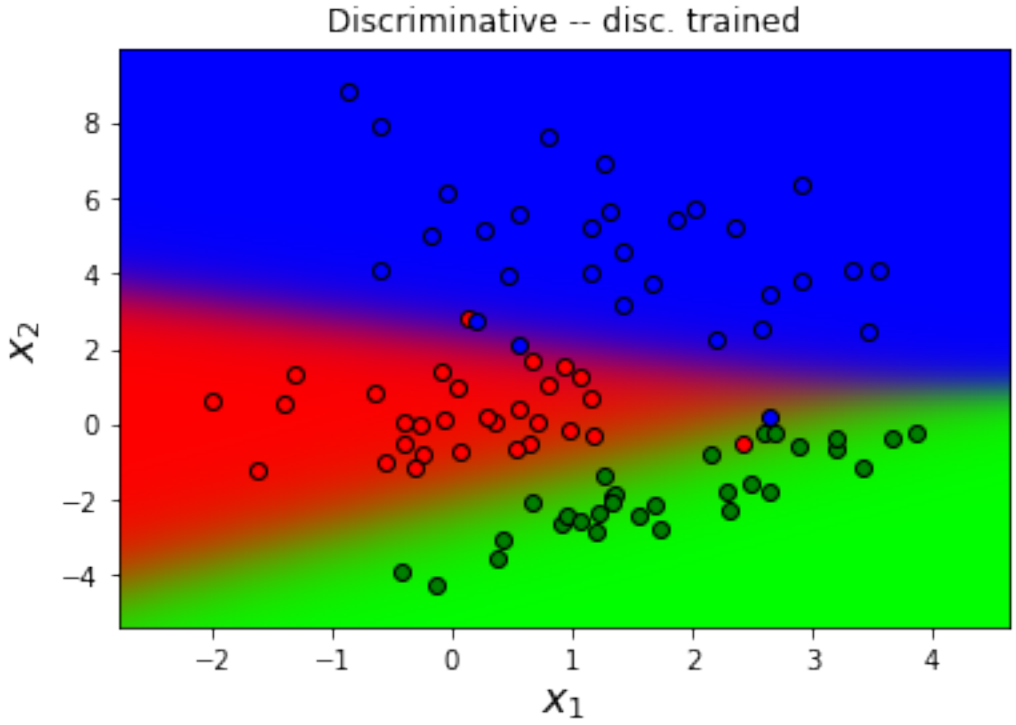


Konečně model dotrénujte diskriminativně. Najděte vhodnou hodnotu learning rate, pro kterou proběhne trénování hezky rychle a hladce.

1 bod

[209]:
```
trained_disc = copy.deepcopy(init_disc)
losses = []

for it_no in range(200):
    loss = trained_disc.gd_step(train_x, train_t, 0.0002)
    losses.append(loss)

plt.figure()
plt.plot(losses)
plt.show()

plot_classifier('Discriminative -- disc. trained', trained_disc, train_samples)
```





Nakonec natrénované klasifikátory srovnejte na neviděných datech, měli byste dosáhnout přesnosti v rozmezí cca 91 % až 98 %.

[205]:
```
classifiers = [
    ('linear  ', linear_classifier),
    ('ml_diag ', classifier_ml_diag_gaussians),
    ('ml_full ', classifier_ml_full_gaussians),
    ('disc_init', init_disc),
    ('disc_ft ', trained_disc),
]

print('model\t\tAcc')
print('-'*22)
for name, classifier in classifiers:
    acc_test = evaluate(classifier, complete_test_x, complete_test_t)
    print(f'{name}\t{100.0*acc_test:.1f} %')
```

```
model           Acc
----------------------
linear          91.1 %
ml_diag         90.0 %
ml_full         94.4 %
disc_init       96.7 %
disc_ft         96.7 %
```

5 Závěrem

Blahopřejeme ke zvládnutí domácí úlohy, snad jste si ji užili. Snad jste si i kladli otázky – proč má GMM občas "zanořené" komponenty? Proč se to stává častěji při Viterbiho trénování? Co se změní, když dotrénujeme lineární klasifikátor diskriminativně? a podobně.

V úloze jsou záměrně nekonzistentní kousky (tvar dat vs. tvar posteriorů, funkce/metody, etc.). Doufáme, že jsme Vás donutili o nich trochu přemýšlet a do života půjdete s názorem na to, který přístup lépe odpovídá Vašemu chápání výpočtů.

Poslední praktická poznámka: Numericky je asi bez výjimky výhodnější provádět výpočty v log-doméně (jako v `ikrlib.py`), takže nedoporučujeme kopírovat bezhlavě snipety odsud do seriózních projektů.