zadani

October 23, 2020

Vítejte u domácí úlohy do SUI. V rámci úlohy Vás čeká několik cvičení, v nichž budete doplňovat poměrně malé fragmenty kódu, místo na ně je vyznačené jako `pass` nebo `None`. Pokud se v buňce s kódem již něco nachází, využijte/neničte to. V dvou případech se očekává textová odpověď, tu uvedete přímo do zadávající buňky. Buňky nerušte ani nepřidávejte.

Maximálně využívejte `numpy` a `torch` pro hromadné operace na celých polích. S výjimkou generátoru minibatchí by se nikde neměl objevit cyklus jdoucí přes jednotlivé příklady.

U všech cvičení je uveden počet bodů za funkční implementaci a orientační počet potřebných řádků. Berte ho prosím opravdu jako orientační, pozornost mu věnujte pouze, pokud ho významně překračujete. Mnoho zdaru!

1 Informace o vzniku řešení

Vyplňte následující údaje **(3 údaje, 0 bodů)**

- Jméno autora: Daniel Konečný
- Login autora: xkonec75
- Datum vzniku: 23. 10. 2020

[1]:
```
import numpy as np
import copy
import matplotlib.pyplot as plt
import scipy.stats
```

2 Přípravné práce

Prvním úkolem v této domácí úloze je načíst data, s nimiž budete pracovat. Vybudujte jednoduchou třídu, která se umí zkonstruovat z cesty k negativním a pozitivním příkladům, a bude poskytovat: - pozitivní a negativní příklady (`dataset.pos`, `dataset.neg` o rozměrech [N, 7]) - všechny příklady a odpovídající třídy (`dataset.xs` o rozměru [N, 7], `dataset.targets` o rozměru [N])

K načítání dat doporučujeme využít `np.loadtxt()`. Netrapte se se zapouzdřování a gettery, berte třídu jako Plain Old Data.

Načtěte trénovací (`{positives,negatives}.trn`), validační (`{positives,negatives}.val`) a testovací (`{positives,negatives}.tst`) dataset, pojmenujte je po řadě (`train_dataset`, `val_dataset`, `test_dataset`).

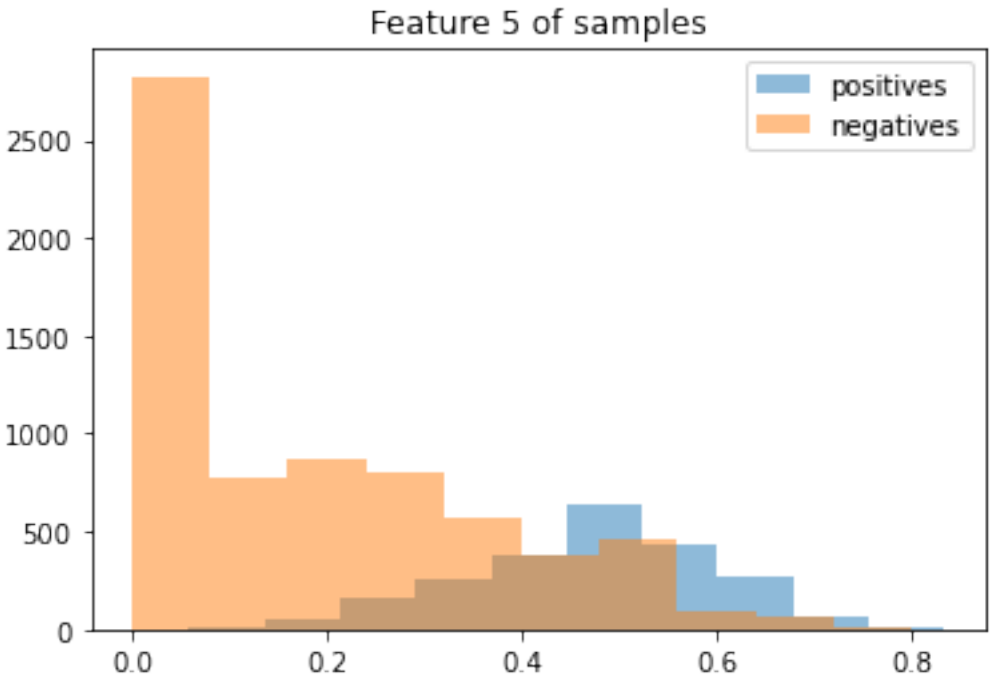
(6+3 řádků, 1 bod)

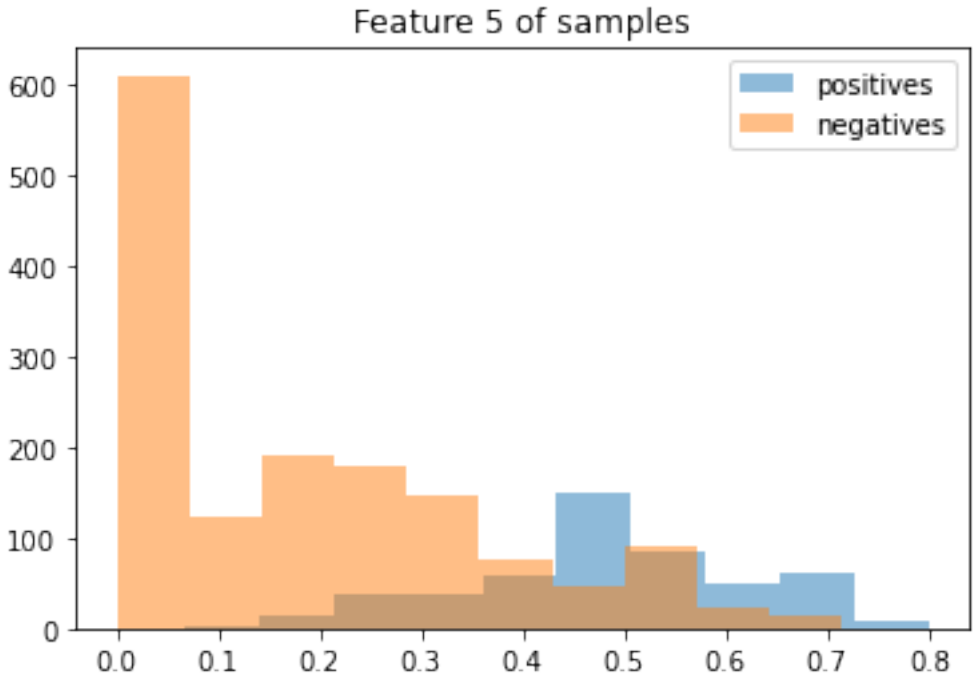
[2]:
```
class Dataset:
    def __init__(self, data_type):
        self.pos = np.loadtxt(f'positives.{data_type}')
        self.neg = np.loadtxt(f'negatives.{data_type}')
        self.xs = np.vstack((self.pos, self.neg))
        self.targets = np.hstack((np.ones(len(self.pos)), np.zeros(len(self.
 ↪neg))))

train_dataset = Dataset('trn')
val_dataset = Dataset('val')
test_dataset = Dataset('tst')

print('positives', train_dataset.pos.shape)
print('negatives', train_dataset.neg.shape)
print('xs', train_dataset.xs.shape)
print('targets', train_dataset.targets.shape)
```

```
positives (2280, 7)
negatives (6841, 7)
xs (9121, 7)
targets (9121,)
```

V řadě následujících cvičení budete pracovat s jedním konkrétním příznakem. Naimplementujte pro začátek funkci, která vykreslí histogram rozložení pozitivních a negativních příkladů (`plt.hist()`). Nezapomeňte na legendu, ať je v grafu jasné, které jsou které. Funkci zavolejte dvakrát, vykreslete histogram příznaku 5 – tzn. šestého ze sedmi – pro trénovací a validační data **(5 řádků, 1 bod)**.

[3]:
```
FOI = 5  # Feature Of Interest

def plot_data(poss, negs):
    plt.hist(poss, alpha=0.5, label='positives')
    plt.hist(negs, alpha=0.5, label='negatives')
    plt.legend()
    plt.title('Feature 5 of samples')
    plt.show()

plot_data(train_dataset.pos[:, FOI], train_dataset.neg[:, FOI])
plot_data(val_dataset.pos[:, FOI], val_dataset.neg[:, FOI])
```


Feature 5 of samples
positives
negatives
2500
2000
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8


Feature 5 of samples
positives
negatives
600
500
400
300
200
100
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

2.0.1 Evaluace klasifikátorů

Než přistoupíte k tvorbě jednotlivých klasifikátorů, vytvořte funkci pro jejich vyhodnocování. Nechť se jmenuje `evaluate` a přijímá po řadě klasifikátor, pole dat (o rozměrech [N] nebo [N, F]) a pole tříd ([N]). Jejím výstupem bude *přesnost*, tzn. podíl správně klasifikovaných příkladů.

Předpokládejte, že klasifikátor poskytuje metodu `.prob_class_1(data)`, která vrací pole posteriorních pravděpodobností třídy 1 (tj. p(y=1|x)) pro daná data. Evaluační funkce bude muset provést tvrdé prahování (na hodnotě 0.5) těchto pravděpodobností a srovnání získaných rozhodnutí s referenčními třídami. Využijte fakt, že `numpy`ovská pole lze mj. porovnávat mezi sebou i se skalárem.

(3 řádky, 1 bod)

[4]:
```
def evaluate(classifier, inputs, targets):
    classification = (classifier.prob_class_1(inputs) > 0.5) == (targets == 1)
    return np.count_nonzero(classification == True) / len(targets)

class Dummy:
    def prob_class_1(self, xs):
        return np.asarray([0.2, 0.7, 0.7])

print(evaluate(Dummy(), None, np.asarray([0, 0, 1])))  # should be 0.66...
```

```
0.6666666666666666
```

2.0.2 Baseline

Vytvořte klasifikátor, který ignoruje vstupní hodnotu dat. Jenom v konstruktoru dostane třídu, kterou má dávat jako tip pro libovolný vstup. Nezapomeňte, že jeho metoda `.prob_class_1(data)` musí vracet pole správné velikosti, využijte `np.ones` nebo `np.full`.

(4 řádky, 1 bod)

[5]:
```
class PriorClassifier:
    def __init__(self, default_class):
        self.default_class = default_class

    def prob_class_1(self, xs):
        return np.full(xs.shape, self.default_class)

baseline = PriorClassifier(0)
val_acc = evaluate(baseline, val_dataset.xs[:, FOI], val_dataset.targets)
print('Baseline val acc:', val_acc)
```

```
Baseline val acc: 0.75
```

3 Generativní klasifikátory

V této části vytvoříte dva generativní klasifikátory, oba založené na Gaussovu rozložení pravděpodobnosti.

Začněte implementací funce, která pro daná 1-D data vrátí Maximum Likelihood odhad střední hodnoty a směrodatné odchylky Gaussova rozložení, které data modeluje. Funkci využijte pro natrénovaní dvou modelů: pozitivních a negativních příkladů. Získané parametry – tzn. střední hodnoty a směrodatné odchylky – vypište.

(5 řádků, 0.5 bodu)

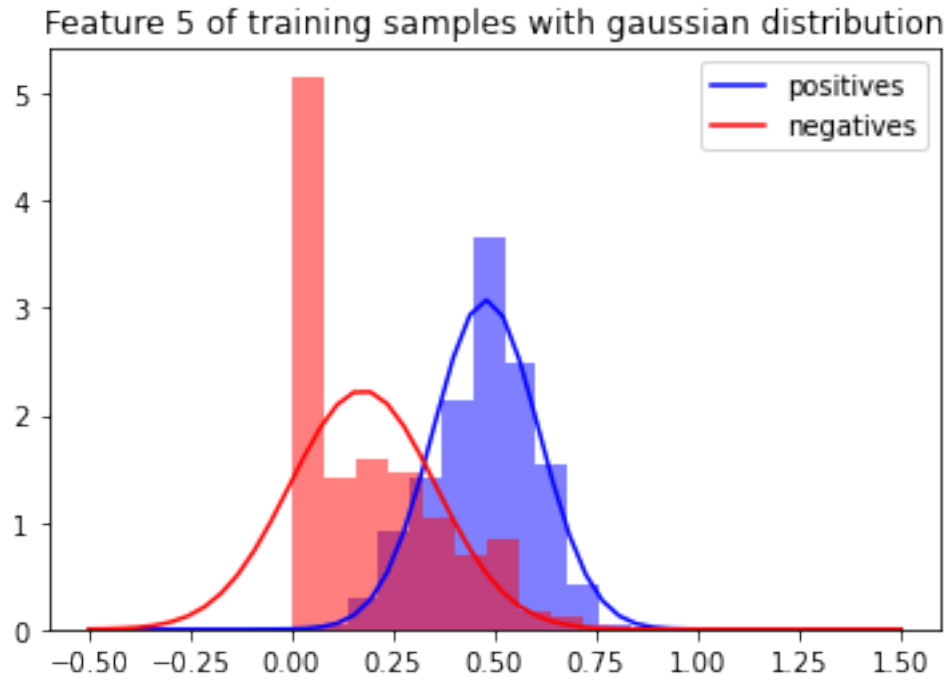
[6]:
```
def get_gauss_params(data):
    return np.mean(data), np.std(data)

pos_mean, pos_std = get_gauss_params(train_dataset.pos[:, FOI])
neg_mean, neg_std = get_gauss_params(train_dataset.neg[:, FOI])
print(f'Positive train samples:\n- mean = {pos_mean:.05f}\n- std = {pos_std:.
 ↪05f}')
print(f'Negative train samples:\n- mean = {neg_mean:.05f}\n- std = {neg_std:.
 ↪05f}')
```

```
Positive train samples:
- mean = 0.47843
- std = 0.12972
Negative train samples:
- mean = 0.17454
- std = 0.17896
```

Ze získaných parametrů vytvořte `scipy`ovská gaussovská rozložení `scipy.stats.norm`. S využitím jejich metody `.pdf()` vytvořte graf, v němž srovnáte skutečné a modelové rozložení pozitivních a negativních příkladů. Rozsah x-ové osy volte od -0.5 do 1.5 (využijte `np.linspace`) a u volání `plt.hist()` nezapomeňte nastavit `density=True`, aby byl histogram normalizovaný a dal se srovnávat s modelem.

(2+8 řádků, 1 bod)

[7]:
```
gauss_pos = scipy.stats.norm(pos_mean, pos_std)
gauss_neg = scipy.stats.norm(neg_mean, neg_std)

axis_range = np.linspace(-0.5, 1.5)
plt.plot(axis_range, gauss_pos.pdf(axis_range), color='blue', label='positives')
plt.plot(axis_range, gauss_neg.pdf(axis_range), color='red', label='negatives')
plt.hist(train_dataset.pos[:, FOI], density=True, alpha=0.5, color='blue')
plt.hist(train_dataset.neg[:, FOI], density=True, alpha=0.5, color='red')
plt.legend()
plt.title('Feature 5 of training samples with gaussian distribution')
plt.show()
```



Naimplementujte binární generativní klasifikátor. Při konstrukci přijímá dvě rozložení poskytující metodu `.pdf()` a odpovídající apriorní pravděpodobnost tříd. Jako všechny klasifikátory v této domácí úloze poskytuje metodu `prob_class_1()`.

(9 řádků, 2 body)

[8]:
```
class BinaryClassifier:
    def __init__(self, classifier0, classifier1, prior):
        self.classifier0 = classifier0
        self.classifier1 = classifier1
        self.p_0 = prior                    # Prior probability of class 0
        self.p_1 = 1 - prior                # Prior probability of class 1

    def prob_class_1(self, xs):
        p_x_0 = self.classifier0.pdf(xs)    # Probability of a sample given␣
↪class 0
        p_x_1 = self.classifier1.pdf(xs)    # Probability of a sample given␣
↪class 1
        p_x = (p_x_0 * self.p_0) + (p_x_1 * self.p_1)   # Probability of a␣
↪sample
        p_1_x = (p_x_1 * self.p_1) / p_x    # Probability of class 1 given a␣
↪sample (Bayes rule)
        return p_1_x
```

Nainstancujte dva generativní klasifikátory: jeden s rovnoměrnými priory a jeden s apriorní pravděpodobností 0.75 pro třídu 0 (negativní příklady). Pomocí funkce `evaluate()` vyhodnoťte

jejich úspěšnost na validačních datech.

(2 řádky, 1 bod)

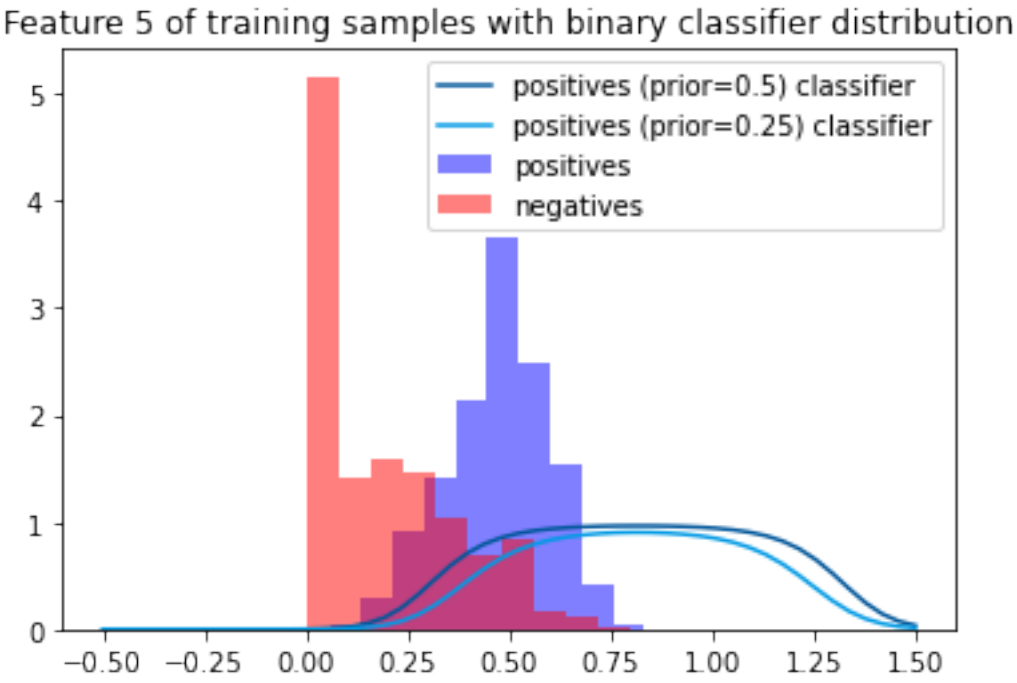
[9]:
```
classifier_flat_prior = BinaryClassifier(gauss_neg, gauss_pos, 0.5)
classifier_full_prior = BinaryClassifier(gauss_neg, gauss_pos, 0.75)

print('flat:', evaluate(classifier_flat_prior, val_dataset.xs[:, FOI],␣
 ↪val_dataset.targets))
print('full:', evaluate(classifier_full_prior, val_dataset.xs[:, FOI],␣
 ↪val_dataset.targets))
```

```
flat: 0.809
full: 0.8475
```

Vykreslete průběh posteriorní pravděpodobnosti třídy 1 jako funkci příznaku 5 pro oba klasifikátory, opět v rozsahu <-0.5; 1.5>. Do grafu zakreslete i histogramy rozložení trénovacích dat, opět s `density=True` pro zachování dynamického rozsahu.

(8 řádků, 1 bod)

[10]:
```
axis_range = np.linspace(-0.5, 1.5)
plt.plot(axis_range, classifier_flat_prior.prob_class_1(axis_range),
         color='#01579B', label='positives (prior=0.5) classifier')
plt.plot(axis_range, classifier_full_prior.prob_class_1(axis_range),
         color='#039BE5', label='positives (prior=0.25) classifier')
plt.hist(train_dataset.pos[:, FOI], density=True, alpha=0.5, color='blue',␣
 ↪label='positives')
plt.hist(train_dataset.neg[:, FOI], density=True, alpha=0.5, color='red',␣
 ↪label='negatives')
plt.legend()
plt.title('Feature 5 of training samples with binary classifier distribution')
plt.show()
```



Interpretujte, přímo v této textové buňce, každou rozhodovací hranici, která je v grafu patrná **(3 věty, 2 body)**: * Rozhodovací hranici určují modré funkce tvořené výstupy binárních klasifikátorů. Pokud jejich hodnota přesáhne 0.5, je vzorek klasifikovaný jako patřící do třídy pozitivních dat. Zvyšováním váhy třídy negativních dat se rozhodovací hranice stává "přísnější" pro data klasifikovaná jako pozitivní – musí být jednoznačněji rozpoznatelná. * Do hodnoty přibližně 0.3 (na ose x) převažují viditelně negativní data, co však už není tak dobře viditelné je, že v hodnotě přibližně 1.3 a dále se začnou data znovu klasifikovat jako negativní (za předpokladu rovných vah). Toto je způsobeno tím, že gaussovy křivky modelující daná data se v hodnotách 0.3 a 1.3 protínají. * První zmíněná rozhodovací hranice se dá také vyčíst z vykreslených histogramů, kde pozitivní začne převyšovat nad negativní kolem hodnoty 0.3.

4 Diskriminativní klasifikátory

V následující části budete přímo modelovat posteriorní pravděpodobnost třídy 1. Modely budou založeny na PyTorchi, ten si prosím nainstalujte. GPU rozhodně nepotřebujete, veškeré výpočty budou velmi rychlé, ne-li bleskové.

Do začátku máte poskytnutou třídu klasifikátoru z jednoho příznaku.

[11]:
```
import torch
import torch.nn.functional as F

class LogisticRegression(torch.nn.Module):
    def __init__(self):
        super().__init__()
```

```
        self.w = torch.nn.parameter.Parameter(torch.tensor([1.0]))
        self.b = torch.nn.parameter.Parameter(torch.tensor([0.0]))

    def forward(self, x):
        return torch.sigmoid(self.w*x + self.b)

    def prob_class_1(self, x):
        prob = self(torch.from_numpy(x))
        return prob.detach().numpy()
```

Pro trénovaní diskriminativních modelů budete potřebovat minibatche. Implementujte funkci, která je bude z daných vstupních a cílových hodnot vytvářet. Výsledkem musí být možno iterovat, ideálně funkci napište jako generátor (využijte klíčové slovo `yield`). Jednotlivé prvky výstupu budou dvojice PyTorchových `FloatTensor`ů (musíte zkonvertovat z numpy a nastavit typ) – první prvek vstupní data, druhý očekávané výstupy. Počítejte s tím, že vstup bude `numpy`ovské pole, rozumná implementace využije `np.random.permutation()` a Advanced Indexing.

Připravený kód funkci použije na konstrukci tří minibatchí pro trénování identity, měli byste vidět celkem pět prvků náhodně uspořádaných do dvojic, ovšem s tím, že s sebou budou mít odpovídající výstupy.

(6 řádků, 2 body)

[12]:
```
def batch_provider(xs, targets, batch_size=10):
    shuffle = np.random.permutation(targets.size)
    for i in range(0, len(shuffle), batch_size):
        batch_indices = shuffle[i:i+batch_size]
        yield torch.from_numpy(xs[batch_indices]).float(), \
              torch.from_numpy(targets[batch_indices]).float()

inputs = np.asarray([1.0, 2.0, 3.0, 4.0, 5.0])
targets = np.asarray([1.0, 2.0, 3.0, 4.0, 5.0])
for x, t in batch_provider(inputs, targets, 2):
    print(f'x: {x}, t: {t}')
```

```
x: tensor([2., 3.]), t: tensor([2., 3.])
x: tensor([5., 1.]), t: tensor([5., 1.])
x: tensor([4.]), t: tensor([4.])
```

Dalším krokem je implementovat funkci, která model vytvoří a natrénuje. Jejím výstupem bude (1) natrénovaný model, (2) průběh trénovací loss a (3) průběh validační přesnosti. Jako model vracejte ten, který dosáhne nejlepší validační přesnosti. Jako loss použijte binární cross-entropii (`F.binary_cross_entropy()`), akumulujte ji přes minibatche a logujte průměr. Pro výpočet validační přesnosti využijte funkci `evaluate()`. Oba průběhy vracejte jako obyčejné seznamy.

V implementaci budete potřebovat dvě zanořené smyčky: jednu pro epochy (průchody přes celý dataset) a uvnitř druhou, která bude iterovat přes jednotlivé minibatche. Na konci každé epochy vyhodnoťte model na validačních datech. K datasetům (trénovacímu a validačnímu) přistupujte bezostyšně jako ke globálním proměnným.

(cca 14 řádků, 3 body)

[13]:
```
def train_llr(model, optimizer, train_xs, val_xs, nb_epochs=100,␣
↪batch_size=256):
    best_model = copy.deepcopy(model)
    losses = []
    accuracies = []

    for epoch_index in range(nb_epochs):
        accumulated_loss = 0
        for batch_x, batch_target in batch_provider(train_xs, train_dataset.
↪targets,batch_size):
            optimizer.zero_grad()
            output = model(batch_x)
            last_loss = F.binary_cross_entropy(output, batch_target)
            last_loss.backward()
            optimizer.step()
            accumulated_loss += last_loss

        last_accuracy = evaluate(model, val_xs, val_dataset.targets)
        print(f'Epoch {epoch_index} - accuracy: {last_accuracy}', end='')
        if all(accuracy < last_accuracy for accuracy in accuracies):
            best_model = copy.deepcopy(model)
            print(' - Model saved.')
        else:
            print('')
        accuracies.append(last_accuracy)
        losses.append(accumulated_loss / (len(train_dataset.xs) // batch_size))

    return best_model, losses, accuracies
```

Funkci zavolejte a natrénujte model. Uveďte zde parametry, které vám dají slušný výsledek. Měli byste dostat přesnost srovnatelnou s generativním klasifikátorem s nastavenými priory. Neměli byste potřebovat víc než 100 epoch. Vykreslete průběh trénovací loss a validační přesnosti, osu x značte v epochách.

V druhém grafu vykreslete histogramy trénovacích dat a pravděpodobnost třídy 1 pro x od -0.5 do 1.5, podobně jako výše u generativních klasifikátorů. Při výpočtu výstupů využijte `with torch.no_grad():`. **(1 + 6 + 9 řádků, 1 bod)**

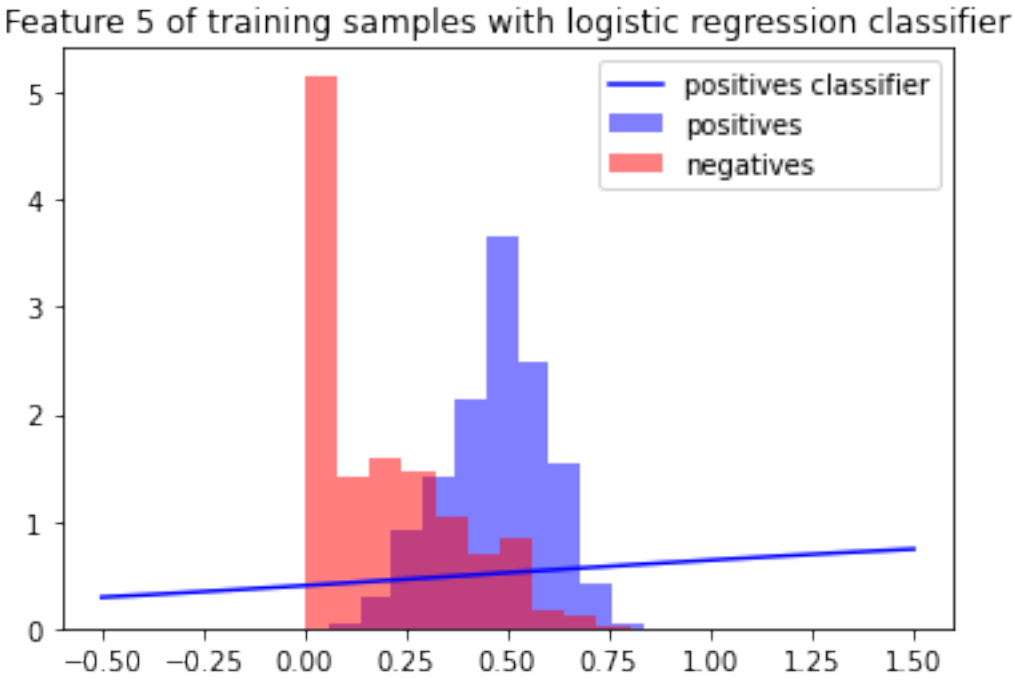
[14]:
```
model = LogisticRegression()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
train_xs = train_dataset.xs[:, FOI]
val_xs = val_dataset.xs[:, FOI]
epoch_count = 70
batch_size = 256

llr_foi_model, losses, accuracies = \
    train_llr(model, optimizer, train_xs, val_xs, epoch_count, batch_size)
```

```
axis_range = np.linspace(0, epoch_count, epoch_count)
plt.plot(axis_range, losses, color='purple', label='loss')
plt.plot(axis_range, accuracies, color='green', label='accuracy')
plt.legend()
plt.title('Training progress')
plt.show()

axis_range = np.linspace(-0.5, 1.5)
llr_foi_model.eval()
with torch.no_grad():
    output = llr_foi_model(axis_range)
plt.plot(axis_range, output, color='blue', label='positives classifier')
plt.hist(train_dataset.pos[:, FOI], density=True, alpha=0.5, color='blue',␣
 ↪label='positives')
plt.hist(train_dataset.neg[:, FOI], density=True, alpha=0.5, color='red',␣
 ↪label='negatives')
plt.legend()
plt.title('Feature 5 of training samples with logistic regression classifier')
plt.show()
```

```
Epoch 0 - accuracy: 0.536 - Model saved.
Epoch 1 - accuracy: 0.536
Epoch 2 - accuracy: 0.536
Epoch 3 - accuracy: 0.5375 - Model saved.
Epoch 4 - accuracy: 0.5435 - Model saved.
Epoch 5 - accuracy: 0.5525 - Model saved.
Epoch 6 - accuracy: 0.5575 - Model saved.
Epoch 7 - accuracy: 0.5625 - Model saved.
Epoch 8 - accuracy: 0.572 - Model saved.
Epoch 9 - accuracy: 0.5835 - Model saved.
Epoch 10 - accuracy: 0.5965 - Model saved.
Epoch 11 - accuracy: 0.6095 - Model saved.
Epoch 12 - accuracy: 0.616 - Model saved.
Epoch 13 - accuracy: 0.6315 - Model saved.
Epoch 14 - accuracy: 0.644 - Model saved.
Epoch 15 - accuracy: 0.659 - Model saved.
Epoch 16 - accuracy: 0.667 - Model saved.
Epoch 17 - accuracy: 0.681 - Model saved.
Epoch 18 - accuracy: 0.683 - Model saved.
Epoch 19 - accuracy: 0.7035 - Model saved.
Epoch 20 - accuracy: 0.7085 - Model saved.
Epoch 21 - accuracy: 0.7185 - Model saved.
Epoch 22 - accuracy: 0.727 - Model saved.
Epoch 23 - accuracy: 0.735 - Model saved.
Epoch 24 - accuracy: 0.763 - Model saved.
Epoch 25 - accuracy: 0.7645 - Model saved.
Epoch 26 - accuracy: 0.774 - Model saved.
```

```
Epoch 27 - accuracy: 0.776 - Model saved.
Epoch 28 - accuracy: 0.7905 - Model saved.
Epoch 29 - accuracy: 0.8 - Model saved.
Epoch 30 - accuracy: 0.804 - Model saved.
Epoch 31 - accuracy: 0.8075 - Model saved.
Epoch 32 - accuracy: 0.809 - Model saved.
Epoch 33 - accuracy: 0.83 - Model saved.
Epoch 34 - accuracy: 0.8305 - Model saved.
Epoch 35 - accuracy: 0.831 - Model saved.
Epoch 36 - accuracy: 0.834 - Model saved.
Epoch 37 - accuracy: 0.836 - Model saved.
Epoch 38 - accuracy: 0.84 - Model saved.
Epoch 39 - accuracy: 0.8425 - Model saved.
Epoch 40 - accuracy: 0.843 - Model saved.
Epoch 41 - accuracy: 0.8475 - Model saved.
Epoch 42 - accuracy: 0.8475
Epoch 43 - accuracy: 0.849 - Model saved.
Epoch 44 - accuracy: 0.8475
Epoch 45 - accuracy: 0.847
Epoch 46 - accuracy: 0.845
Epoch 47 - accuracy: 0.8435
Epoch 48 - accuracy: 0.844
Epoch 49 - accuracy: 0.844
Epoch 50 - accuracy: 0.842
Epoch 51 - accuracy: 0.84
Epoch 52 - accuracy: 0.84
Epoch 53 - accuracy: 0.84
Epoch 54 - accuracy: 0.8305
Epoch 55 - accuracy: 0.8305
Epoch 56 - accuracy: 0.8305
Epoch 57 - accuracy: 0.8305
Epoch 58 - accuracy: 0.8285
Epoch 59 - accuracy: 0.8275
Epoch 60 - accuracy: 0.825
Epoch 61 - accuracy: 0.815
Epoch 62 - accuracy: 0.807
Epoch 63 - accuracy: 0.8055
Epoch 64 - accuracy: 0.8055
Epoch 65 - accuracy: 0.7945
Epoch 66 - accuracy: 0.793
Epoch 67 - accuracy: 0.791
Epoch 68 - accuracy: 0.7905
Epoch 69 - accuracy: 0.785
```


Training progress
loss
accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
10
20
30
40
50
60
70


Feature 5 of training samples with logistic regression classifier
positives classifier
positives
negatives
5
4
3
2
1
0
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50

4.1 Všechny vstupní příznaky

V posledním cvičení natrénujete logistickou regresi, která využije všech sedm vstupních příznaků.

Prvním krokem je naimplementovat příslusný model. Bezostyšně zkopírujte tělo třídy `LogisticRegresion` a upravte ji tak, aby zvládala libovolný počet vstupů, využijte `torch.nn.Linear`. U výstupu metody `.forward()` dejte pozor, aby měl výstup tvar `[N]`; pravděpodobně budete potřebovat `squeeze`.

(9 řádků, 1 bod)

[15]:
```
class LogisticRegressionND(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = torch.nn.Linear(7, 1)

    def forward(self, x):
        return np.squeeze(torch.sigmoid(self.linear(x)))

    def prob_class_1(self, x):
        prob = self(torch.from_numpy(x).float())
        return prob.detach().numpy()
```

Podobně jako u jednodimenzionálni regrese implementujte funkci pro trénovaní plné logistické regrese. V ideálním případě vyfaktorujete společnou implementaci, které budete pouze předávat různá trénovací a validační data.

Zvídaví mohou zkusit `Adam`a jako optimalizátor namísto obyčejného `SGD`.

Funkci zavolejte, natrénujte model. Opět vykreslete průběh trénovací loss a validační přesnosti. Měli byste se s přesností dostat nad 90 %.

(ne víc než cca 30 řádků při kopírování, 1 bod)

[16]:
```
# Implementation factorized, see above.

model = LogisticRegressionND()
optimizer = torch.optim.Adam(model.parameters())
train_xs = train_dataset.xs
val_xs = val_dataset.xs
epoch_count = 400
batch_size = 256

llr_full_model, losses, accuracies = \
    train_llr(model, optimizer, train_xs, val_xs, epoch_count, batch_size)

axis_range = np.linspace(0, epoch_count, epoch_count)
plt.plot(axis_range, losses, color='purple', label='loss')
plt.plot(axis_range, accuracies, color='green', label='accuracy')
plt.legend()
plt.title('Training progress')
```

```
plt.show()
```

```
Epoch 0 - accuracy: 0.7325 - Model saved.
Epoch 1 - accuracy: 0.7055
Epoch 2 - accuracy: 0.6735
Epoch 3 - accuracy: 0.6125
Epoch 4 - accuracy: 0.5615
Epoch 5 - accuracy: 0.5855
Epoch 6 - accuracy: 0.6285
Epoch 7 - accuracy: 0.6685
Epoch 8 - accuracy: 0.7165
Epoch 9 - accuracy: 0.7605 - Model saved.
Epoch 10 - accuracy: 0.812 - Model saved.
Epoch 11 - accuracy: 0.856 - Model saved.
Epoch 12 - accuracy: 0.897 - Model saved.
Epoch 13 - accuracy: 0.925 - Model saved.
Epoch 14 - accuracy: 0.9355 - Model saved.
Epoch 15 - accuracy: 0.948 - Model saved.
Epoch 16 - accuracy: 0.9505 - Model saved.
Epoch 17 - accuracy: 0.953 - Model saved.
Epoch 18 - accuracy: 0.956 - Model saved.
Epoch 19 - accuracy: 0.9575 - Model saved.
Epoch 20 - accuracy: 0.957
Epoch 21 - accuracy: 0.9565
Epoch 22 - accuracy: 0.9575
Epoch 23 - accuracy: 0.9575
Epoch 24 - accuracy: 0.958 - Model saved.
Epoch 25 - accuracy: 0.9585 - Model saved.
Epoch 26 - accuracy: 0.9585
Epoch 27 - accuracy: 0.958
Epoch 28 - accuracy: 0.958
Epoch 29 - accuracy: 0.958
Epoch 30 - accuracy: 0.9585
Epoch 31 - accuracy: 0.958
Epoch 32 - accuracy: 0.9585
Epoch 33 - accuracy: 0.959 - Model saved.
Epoch 34 - accuracy: 0.9595 - Model saved.
Epoch 35 - accuracy: 0.96 - Model saved.
Epoch 36 - accuracy: 0.961 - Model saved.
Epoch 37 - accuracy: 0.961
Epoch 38 - accuracy: 0.9615 - Model saved.
Epoch 39 - accuracy: 0.962 - Model saved.
Epoch 40 - accuracy: 0.9625 - Model saved.
Epoch 41 - accuracy: 0.9625
Epoch 42 - accuracy: 0.9625
Epoch 43 - accuracy: 0.962
Epoch 44 - accuracy: 0.9635 - Model saved.
Epoch 45 - accuracy: 0.9625
```

```
Epoch 46 - accuracy: 0.964 - Model saved.
Epoch 47 - accuracy: 0.964
Epoch 48 - accuracy: 0.9645 - Model saved.
Epoch 49 - accuracy: 0.965 - Model saved.
Epoch 50 - accuracy: 0.965
Epoch 51 - accuracy: 0.965
Epoch 52 - accuracy: 0.965
Epoch 53 - accuracy: 0.965
Epoch 54 - accuracy: 0.965
Epoch 55 - accuracy: 0.9645
Epoch 56 - accuracy: 0.9645
Epoch 57 - accuracy: 0.9645
Epoch 58 - accuracy: 0.9645
Epoch 59 - accuracy: 0.965
Epoch 60 - accuracy: 0.9665 - Model saved.
Epoch 61 - accuracy: 0.9665
Epoch 62 - accuracy: 0.9665
Epoch 63 - accuracy: 0.9675 - Model saved.
Epoch 64 - accuracy: 0.9675
Epoch 65 - accuracy: 0.9665
Epoch 66 - accuracy: 0.9665
Epoch 67 - accuracy: 0.968 - Model saved.
Epoch 68 - accuracy: 0.9685 - Model saved.
Epoch 69 - accuracy: 0.968
Epoch 70 - accuracy: 0.9685
Epoch 71 - accuracy: 0.968
Epoch 72 - accuracy: 0.9685
Epoch 73 - accuracy: 0.9685
Epoch 74 - accuracy: 0.969 - Model saved.
Epoch 75 - accuracy: 0.9685
Epoch 76 - accuracy: 0.969
Epoch 77 - accuracy: 0.9705 - Model saved.
Epoch 78 - accuracy: 0.9705
Epoch 79 - accuracy: 0.971 - Model saved.
Epoch 80 - accuracy: 0.971
Epoch 81 - accuracy: 0.971
Epoch 82 - accuracy: 0.9715 - Model saved.
Epoch 83 - accuracy: 0.9715
Epoch 84 - accuracy: 0.9715
Epoch 85 - accuracy: 0.971
Epoch 86 - accuracy: 0.9715
Epoch 87 - accuracy: 0.971
Epoch 88 - accuracy: 0.9715
Epoch 89 - accuracy: 0.971
Epoch 90 - accuracy: 0.971
Epoch 91 - accuracy: 0.9705
Epoch 92 - accuracy: 0.9705
Epoch 93 - accuracy: 0.9705
```

```
Epoch 94 - accuracy: 0.971
Epoch 95 - accuracy: 0.9705
Epoch 96 - accuracy: 0.9705
Epoch 97 - accuracy: 0.971
Epoch 98 - accuracy: 0.9705
Epoch 99 - accuracy: 0.9705
Epoch 100 - accuracy: 0.9705
Epoch 101 - accuracy: 0.9705
Epoch 102 - accuracy: 0.9705
Epoch 103 - accuracy: 0.9705
Epoch 104 - accuracy: 0.9705
Epoch 105 - accuracy: 0.9705
Epoch 106 - accuracy: 0.9705
Epoch 107 - accuracy: 0.9705
Epoch 108 - accuracy: 0.9715
Epoch 109 - accuracy: 0.9705
Epoch 110 - accuracy: 0.9705
Epoch 111 - accuracy: 0.9705
Epoch 112 - accuracy: 0.9715
Epoch 113 - accuracy: 0.971
Epoch 114 - accuracy: 0.971
Epoch 115 - accuracy: 0.971
Epoch 116 - accuracy: 0.9715
Epoch 117 - accuracy: 0.971
Epoch 118 - accuracy: 0.971
Epoch 119 - accuracy: 0.9705
Epoch 120 - accuracy: 0.9715
Epoch 121 - accuracy: 0.971
Epoch 122 - accuracy: 0.9715
Epoch 123 - accuracy: 0.971
Epoch 124 - accuracy: 0.972 - Model saved.
Epoch 125 - accuracy: 0.9715
Epoch 126 - accuracy: 0.9715
Epoch 127 - accuracy: 0.9725 - Model saved.
Epoch 128 - accuracy: 0.9715
Epoch 129 - accuracy: 0.9715
Epoch 130 - accuracy: 0.9715
Epoch 131 - accuracy: 0.9715
Epoch 132 - accuracy: 0.972
Epoch 133 - accuracy: 0.972
Epoch 134 - accuracy: 0.972
Epoch 135 - accuracy: 0.972
Epoch 136 - accuracy: 0.9725
Epoch 137 - accuracy: 0.9725
Epoch 138 - accuracy: 0.9725
Epoch 139 - accuracy: 0.9725
Epoch 140 - accuracy: 0.974 - Model saved.
Epoch 141 - accuracy: 0.9735
```

```
Epoch 142 - accuracy: 0.973
Epoch 143 - accuracy: 0.973
Epoch 144 - accuracy: 0.9735
Epoch 145 - accuracy: 0.9735
Epoch 146 - accuracy: 0.974
Epoch 147 - accuracy: 0.9735
Epoch 148 - accuracy: 0.974
Epoch 149 - accuracy: 0.9735
Epoch 150 - accuracy: 0.974
Epoch 151 - accuracy: 0.9735
Epoch 152 - accuracy: 0.974
Epoch 153 - accuracy: 0.975 - Model saved.
Epoch 154 - accuracy: 0.974
Epoch 155 - accuracy: 0.975
Epoch 156 - accuracy: 0.975
Epoch 157 - accuracy: 0.9755 - Model saved.
Epoch 158 - accuracy: 0.9735
Epoch 159 - accuracy: 0.975
Epoch 160 - accuracy: 0.975
Epoch 161 - accuracy: 0.975
Epoch 162 - accuracy: 0.975
Epoch 163 - accuracy: 0.975
Epoch 164 - accuracy: 0.976 - Model saved.
Epoch 165 - accuracy: 0.9745
Epoch 166 - accuracy: 0.976
Epoch 167 - accuracy: 0.976
Epoch 168 - accuracy: 0.976
Epoch 169 - accuracy: 0.975
Epoch 170 - accuracy: 0.9755
Epoch 171 - accuracy: 0.976
Epoch 172 - accuracy: 0.9755
Epoch 173 - accuracy: 0.9755
Epoch 174 - accuracy: 0.976
Epoch 175 - accuracy: 0.9765 - Model saved.
Epoch 176 - accuracy: 0.9745
Epoch 177 - accuracy: 0.976
Epoch 178 - accuracy: 0.9765
Epoch 179 - accuracy: 0.975
Epoch 180 - accuracy: 0.976
Epoch 181 - accuracy: 0.9765
Epoch 182 - accuracy: 0.976
Epoch 183 - accuracy: 0.976
Epoch 184 - accuracy: 0.976
Epoch 185 - accuracy: 0.976
Epoch 186 - accuracy: 0.9755
Epoch 187 - accuracy: 0.976
Epoch 188 - accuracy: 0.976
Epoch 189 - accuracy: 0.9765
```

```
Epoch 190 - accuracy: 0.9765
Epoch 191 - accuracy: 0.974
Epoch 192 - accuracy: 0.976
Epoch 193 - accuracy: 0.9765
Epoch 194 - accuracy: 0.9765
Epoch 195 - accuracy: 0.9765
Epoch 196 - accuracy: 0.976
Epoch 197 - accuracy: 0.976
Epoch 198 - accuracy: 0.976
Epoch 199 - accuracy: 0.976
Epoch 200 - accuracy: 0.974
Epoch 201 - accuracy: 0.976
Epoch 202 - accuracy: 0.9745
Epoch 203 - accuracy: 0.975
Epoch 204 - accuracy: 0.976
Epoch 205 - accuracy: 0.976
Epoch 206 - accuracy: 0.976
Epoch 207 - accuracy: 0.9755
Epoch 208 - accuracy: 0.976
Epoch 209 - accuracy: 0.9755
Epoch 210 - accuracy: 0.976
Epoch 211 - accuracy: 0.976
Epoch 212 - accuracy: 0.975
Epoch 213 - accuracy: 0.9765
Epoch 214 - accuracy: 0.9755
Epoch 215 - accuracy: 0.9765
Epoch 216 - accuracy: 0.9775 - Model saved.
Epoch 217 - accuracy: 0.9765
Epoch 218 - accuracy: 0.9755
Epoch 219 - accuracy: 0.9755
Epoch 220 - accuracy: 0.9765
Epoch 221 - accuracy: 0.976
Epoch 222 - accuracy: 0.9765
Epoch 223 - accuracy: 0.976
Epoch 224 - accuracy: 0.977
Epoch 225 - accuracy: 0.976
Epoch 226 - accuracy: 0.977
Epoch 227 - accuracy: 0.9765
Epoch 228 - accuracy: 0.977
Epoch 229 - accuracy: 0.976
Epoch 230 - accuracy: 0.9765
Epoch 231 - accuracy: 0.977
Epoch 232 - accuracy: 0.9755
Epoch 233 - accuracy: 0.9765
Epoch 234 - accuracy: 0.9765
Epoch 235 - accuracy: 0.976
Epoch 236 - accuracy: 0.9765
Epoch 237 - accuracy: 0.9775
```

```
Epoch 238 - accuracy: 0.976
Epoch 239 - accuracy: 0.978 - Model saved.
Epoch 240 - accuracy: 0.9785 - Model saved.
Epoch 241 - accuracy: 0.976
Epoch 242 - accuracy: 0.976
Epoch 243 - accuracy: 0.976
Epoch 244 - accuracy: 0.9775
Epoch 245 - accuracy: 0.976
Epoch 246 - accuracy: 0.977
Epoch 247 - accuracy: 0.9755
Epoch 248 - accuracy: 0.978
Epoch 249 - accuracy: 0.9775
Epoch 250 - accuracy: 0.9785
Epoch 251 - accuracy: 0.9775
Epoch 252 - accuracy: 0.9775
Epoch 253 - accuracy: 0.9765
Epoch 254 - accuracy: 0.9785
Epoch 255 - accuracy: 0.9775
Epoch 256 - accuracy: 0.978
Epoch 257 - accuracy: 0.976
Epoch 258 - accuracy: 0.9785
Epoch 259 - accuracy: 0.9765
Epoch 260 - accuracy: 0.9775
Epoch 261 - accuracy: 0.978
Epoch 262 - accuracy: 0.977
Epoch 263 - accuracy: 0.976
Epoch 264 - accuracy: 0.9785
Epoch 265 - accuracy: 0.977
Epoch 266 - accuracy: 0.977
Epoch 267 - accuracy: 0.977
Epoch 268 - accuracy: 0.977
Epoch 269 - accuracy: 0.9765
Epoch 270 - accuracy: 0.9785
Epoch 271 - accuracy: 0.978
Epoch 272 - accuracy: 0.978
Epoch 273 - accuracy: 0.9785
Epoch 274 - accuracy: 0.9765
Epoch 275 - accuracy: 0.9775
Epoch 276 - accuracy: 0.9765
Epoch 277 - accuracy: 0.9785
Epoch 278 - accuracy: 0.9765
Epoch 279 - accuracy: 0.9775
Epoch 280 - accuracy: 0.978
Epoch 281 - accuracy: 0.978
Epoch 282 - accuracy: 0.978
Epoch 283 - accuracy: 0.9775
Epoch 284 - accuracy: 0.978
Epoch 285 - accuracy: 0.978
```

```
Epoch 286 - accuracy: 0.9775
Epoch 287 - accuracy: 0.978
Epoch 288 - accuracy: 0.978
Epoch 289 - accuracy: 0.9775
Epoch 290 - accuracy: 0.979 - Model saved.
Epoch 291 - accuracy: 0.979
Epoch 292 - accuracy: 0.9775
Epoch 293 - accuracy: 0.978
Epoch 294 - accuracy: 0.979
Epoch 295 - accuracy: 0.978
Epoch 296 - accuracy: 0.978
Epoch 297 - accuracy: 0.978
Epoch 298 - accuracy: 0.979
Epoch 299 - accuracy: 0.977
Epoch 300 - accuracy: 0.978
Epoch 301 - accuracy: 0.978
Epoch 302 - accuracy: 0.977
Epoch 303 - accuracy: 0.978
Epoch 304 - accuracy: 0.978
Epoch 305 - accuracy: 0.978
Epoch 306 - accuracy: 0.979
Epoch 307 - accuracy: 0.979
Epoch 308 - accuracy: 0.979
Epoch 309 - accuracy: 0.979
Epoch 310 - accuracy: 0.9775
Epoch 311 - accuracy: 0.978
Epoch 312 - accuracy: 0.978
Epoch 313 - accuracy: 0.978
Epoch 314 - accuracy: 0.978
Epoch 315 - accuracy: 0.9785
Epoch 316 - accuracy: 0.978
Epoch 317 - accuracy: 0.9775
Epoch 318 - accuracy: 0.978
Epoch 319 - accuracy: 0.978
Epoch 320 - accuracy: 0.978
Epoch 321 - accuracy: 0.978
Epoch 322 - accuracy: 0.9775
Epoch 323 - accuracy: 0.9785
Epoch 324 - accuracy: 0.9775
Epoch 325 - accuracy: 0.978
Epoch 326 - accuracy: 0.978
Epoch 327 - accuracy: 0.978
Epoch 328 - accuracy: 0.978
Epoch 329 - accuracy: 0.978
Epoch 330 - accuracy: 0.978
Epoch 331 - accuracy: 0.9785
Epoch 332 - accuracy: 0.978
Epoch 333 - accuracy: 0.978
```

```
Epoch 334 - accuracy: 0.9775
Epoch 335 - accuracy: 0.978
Epoch 336 - accuracy: 0.9775
Epoch 337 - accuracy: 0.978
Epoch 338 - accuracy: 0.9775
Epoch 339 - accuracy: 0.978
Epoch 340 - accuracy: 0.9775
Epoch 341 - accuracy: 0.9775
Epoch 342 - accuracy: 0.9775
Epoch 343 - accuracy: 0.9785
Epoch 344 - accuracy: 0.9775
Epoch 345 - accuracy: 0.978
Epoch 346 - accuracy: 0.9775
Epoch 347 - accuracy: 0.978
Epoch 348 - accuracy: 0.9775
Epoch 349 - accuracy: 0.9775
Epoch 350 - accuracy: 0.978
Epoch 351 - accuracy: 0.978
Epoch 352 - accuracy: 0.9775
Epoch 353 - accuracy: 0.978
Epoch 354 - accuracy: 0.9775
Epoch 355 - accuracy: 0.978
Epoch 356 - accuracy: 0.978
Epoch 357 - accuracy: 0.9775
Epoch 358 - accuracy: 0.978
Epoch 359 - accuracy: 0.9775
Epoch 360 - accuracy: 0.977
Epoch 361 - accuracy: 0.978
Epoch 362 - accuracy: 0.9775
Epoch 363 - accuracy: 0.977
Epoch 364 - accuracy: 0.9775
Epoch 365 - accuracy: 0.978
Epoch 366 - accuracy: 0.978
Epoch 367 - accuracy: 0.977
Epoch 368 - accuracy: 0.977
Epoch 369 - accuracy: 0.977
Epoch 370 - accuracy: 0.977
Epoch 371 - accuracy: 0.9775
Epoch 372 - accuracy: 0.977
Epoch 373 - accuracy: 0.9775
Epoch 374 - accuracy: 0.9775
Epoch 375 - accuracy: 0.9775
Epoch 376 - accuracy: 0.9775
Epoch 377 - accuracy: 0.9775
Epoch 378 - accuracy: 0.9765
Epoch 379 - accuracy: 0.977
Epoch 380 - accuracy: 0.9775
Epoch 381 - accuracy: 0.977
```

```
Epoch 382 - accuracy: 0.9775
Epoch 383 - accuracy: 0.978
Epoch 384 - accuracy: 0.9765
Epoch 385 - accuracy: 0.9765
Epoch 386 - accuracy: 0.978
Epoch 387 - accuracy: 0.9765
Epoch 388 - accuracy: 0.9775
Epoch 389 - accuracy: 0.9775
Epoch 390 - accuracy: 0.9775
Epoch 391 - accuracy: 0.9765
Epoch 392 - accuracy: 0.9775
Epoch 393 - accuracy: 0.9775
Epoch 394 - accuracy: 0.9765
Epoch 395 - accuracy: 0.977
Epoch 396 - accuracy: 0.9765
Epoch 397 - accuracy: 0.9775
Epoch 398 - accuracy: 0.977
Epoch 399 - accuracy: 0.977
```

5 Závěrem

Konečně vyhodnoťte všech pět vytvořených klasifikátorů na testovacích datech. Stačí doplnit jejich názvy a předat jim příznaky, na které jsou zvyklé.

(0.5 bodu)

[17]:
```
xs_full = test_dataset.xs
xs_foi = test_dataset.xs[:, FOI]
targets = test_dataset.targets

print('Baseline:', evaluate(baseline, xs_foi, targets))
print('Generative classifier (w/o prior):', evaluate(classifier_flat_prior,␣
 ↪xs_foi, targets))
print('Generative classifier (correct):', evaluate(classifier_full_prior,␣
 ↪xs_foi, targets))
print('Logistic regression:', evaluate(llr_foi_model, xs_foi, targets))
print('Logistic regression all features:', evaluate(llr_full_model, xs_full,␣
 ↪targets))
```

```
Baseline: 0.75
Generative classifier (w/o prior): 0.8
Generative classifier (correct): 0.847
Logistic regression: 0.847
Logistic regression all features: 0.9675
```

Blahopřejeme ke zvládnutí domácí úlohy! Notebook spusťte načisto (Kernel -> Restart & Run all), vyexportuje jako PDF a odevzdejte pojmenovaný svým loginem.

Mimochodem, vstupní data nejsou synteticky generovaná. Nasbírali jsme je z projektu; Vaše klasifikátory v této domácí úloze predikují, že daný hráč vyhraje; takže by se daly použít jako heuristika pro ohodnocování listových uzlů ve stavovém prostoru hry. Pro představu, odhadujete to z pozic pět kol před koncem partie pro daného hráče. Poskytnuté příznaky popisují globální charakteristiky stavu hry jako je například poměr délky hranic předmětného hráče k ostatním hranicím.