Vítejte u prvního projektu do SUI. V rámci projektu Vás čeká několik cvičení, v nichž budete doplňovat poměrně malé fragmenty kódu (místo je vyznačeno pomocí `None` nebo `pass`). Pokud se v buňce s kódem již něco nachází, využijte/neničte to. Buňky nerušte ani nepřidávejte. Snažte se programovat hezky, ale jediná skutečně aktivně zakázaná, vyhledávaná a -- i opakovaně -- postihovaná technika je cyklení přes data (ať už explicitním cyklem nebo v rámci `list`/`dict` comprehension), tomu se vyhýbejte jako čert kříží a řešte to pomocí vhodných operací lineární algebry.

Až budete s řešením hotovi, vyexportujte ho ("Download as") jako PDF i pythonovský skript a ty odevzdejte **pojmenované názvem týmu** (tj. loginem vedoucího). Dbejte, aby bylo v PDF všechno vidět (nezůstal kód za okrajem stránky apod.).

U všech cvičení je uveden orientační počet řádků řešení. Berte ho prosím opravdu jako orientační, pozornost mu věnujte, pouze pokud ho významně překračujete.

```
import numpy as np
import copy
import matplotlib.pyplot as plt
import scipy.stats
```

Přípravné práce

Prvním úkolem v tomto projektu 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 zapouzdřováním 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` a `test_dataset`.

(6 řádků)

```
class BinaryDataset:
    def __init__(self, filePathPos: str, filePathNeg: str):
        self.pos = np.loadtxt(filePathPos, dtype=float, delimiter=' ')
        self.neg = np.loadtxt(filePathNeg, dtype=float, delimiter=' ')
        self.xs = np.concatenate((self.pos, self.neg))
        self.targets = np.concatenate((np.ones(self.pos.shape[0]),
np.zeros(self.neg.shape[0])))

train_dataset = BinaryDataset('positives.trn', 'negatives.trn')
val_dataset = BinaryDataset('positives.val', 'negatives.val')
```

```
test_dataset = BinaryDataset('positives.tst', 'negatives.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,)
```

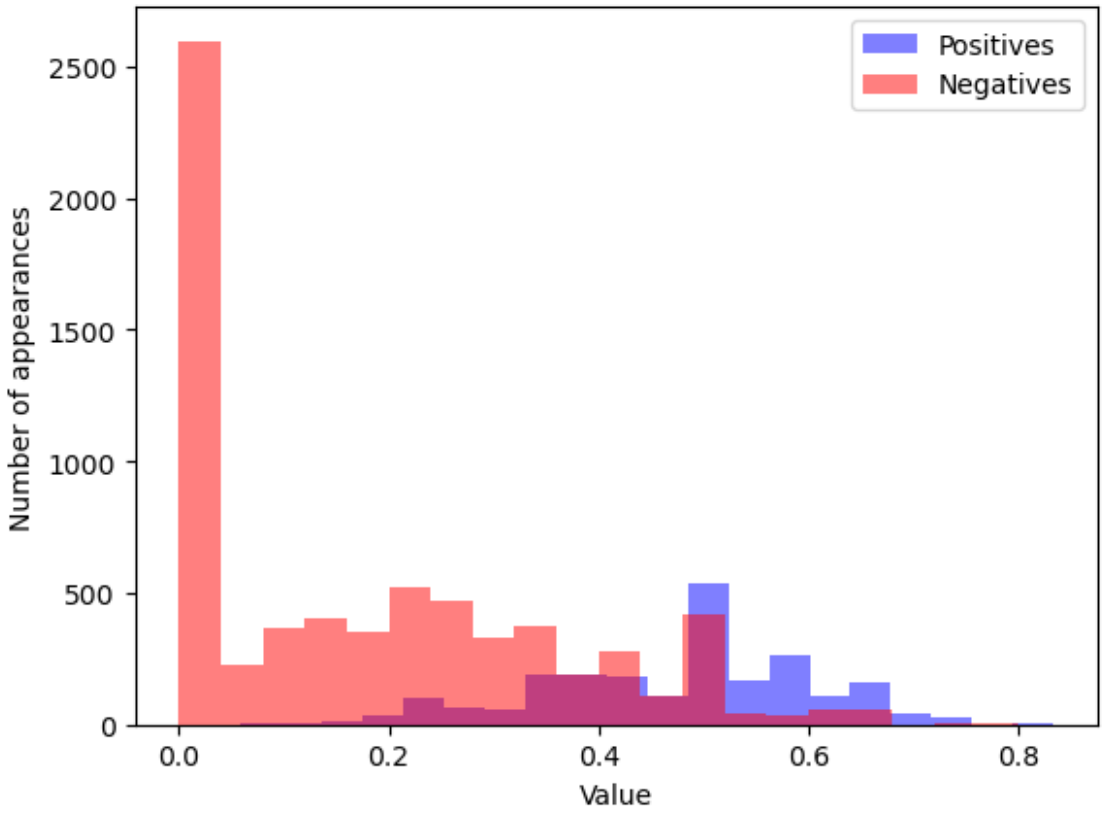
V řadě následujících cvičení budete pracovat s jedním konkrétním příznakem. Naimplementujte proto funkci, která vykreslí histogram rozložení pozitivních a negativních příkladů z jedné sady. Nezapomeňte na legendu, ať je v grafu jasné, které jsou které. Funkci zavoláte dvakrát, vykreslete histogram příznaku `5` -- tzn. šestého ze sedmi -- pro trénovací a validační data

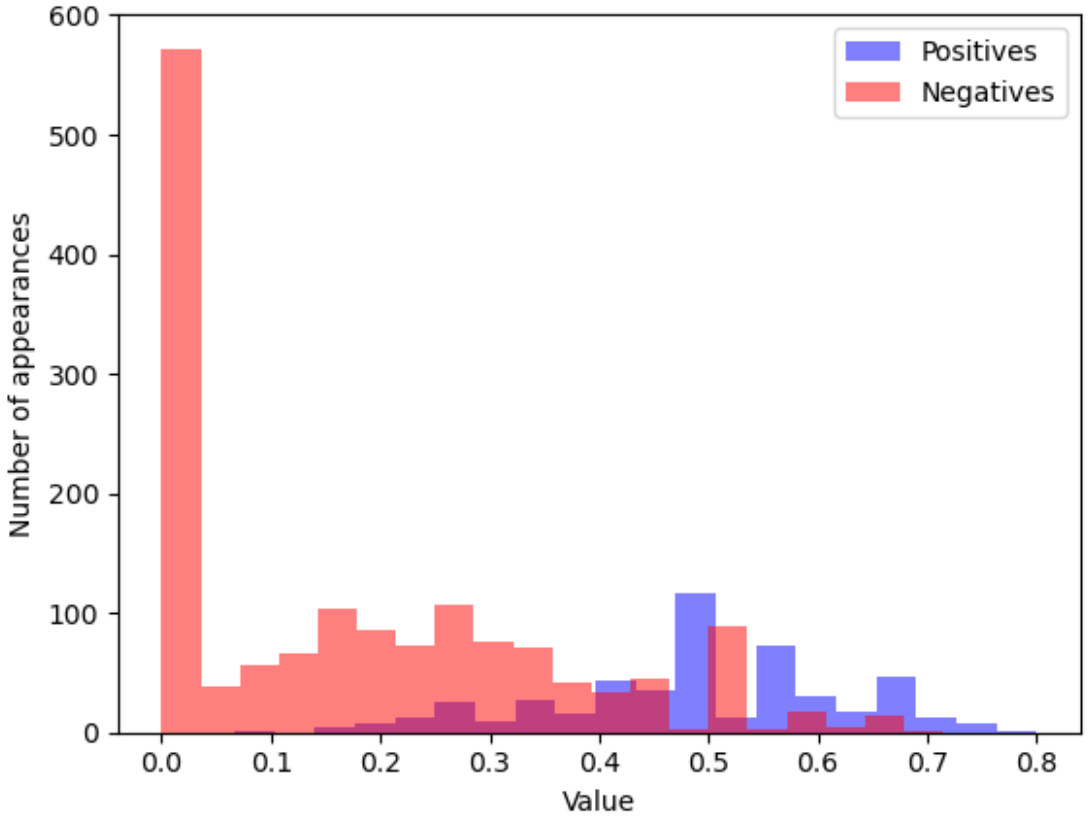
(5 řádků)

```
FOI = 5  # Feature Of Interest

def plot_data(poss, negs):
    plt.hist(poss, bins=20, alpha=0.5, label='Positives', color='b')
    plt.hist(negs, bins=20,  alpha=0.5, label='Negatives', color='r')
    plt.legend()
    plt.xlabel('Value')
    plt.ylabel('Number of appearances')
    plt.show()

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


Positives
Negatives
Number of appearances
2500
2000
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8
Value


Positives
Negatives
Number of appearances
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
Value

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, F]) a pole tříd ([N]). Jejím výstupem bude *přesnost* (accuracy), tzn. podíl správně klasifikovaných příkladů.

Předpokládejte, že klasifikátor poskytuje metodu `.prob_class_1(data)`, která vrací pole posteriorních pravděpodobností třídy 1 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 se skalárem.

(3 řádky)

```
def evaluate(classifier, inputs, targets):
    decisions = classifier.prob_class_1(inputs) > 0.5
    correctDecisions = (decisions == targets)
    return np.mean(correctDecisions)


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
```

Baseline

Vytvořte klasifikátor, který ignoruje vstupní data. 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.

(4 řádky)

```
class PriorClassifier:
    def __init__(self, value):
        self.value = value
    def prob_class_1(self, xs):
        return np.full(xs.shape[0], self.value)

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
```

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 -- vypíšete.

(1 řádek)

```
def mle_gauss_1d(data):
    return np.mean(data), np.std(data)

mu_pos, std_pos = mle_gauss_1d(train_dataset.pos[:, FOI])
mu_neg, std_neg = mle_gauss_1d(train_dataset.neg[:, FOI])

print('Pos mean: {:.2f} std: {:.2f}'.format(mu_pos, std_pos))
print('Neg mean: {:.2f} std: {:.2f}'.format(mu_neg, std_neg))
```

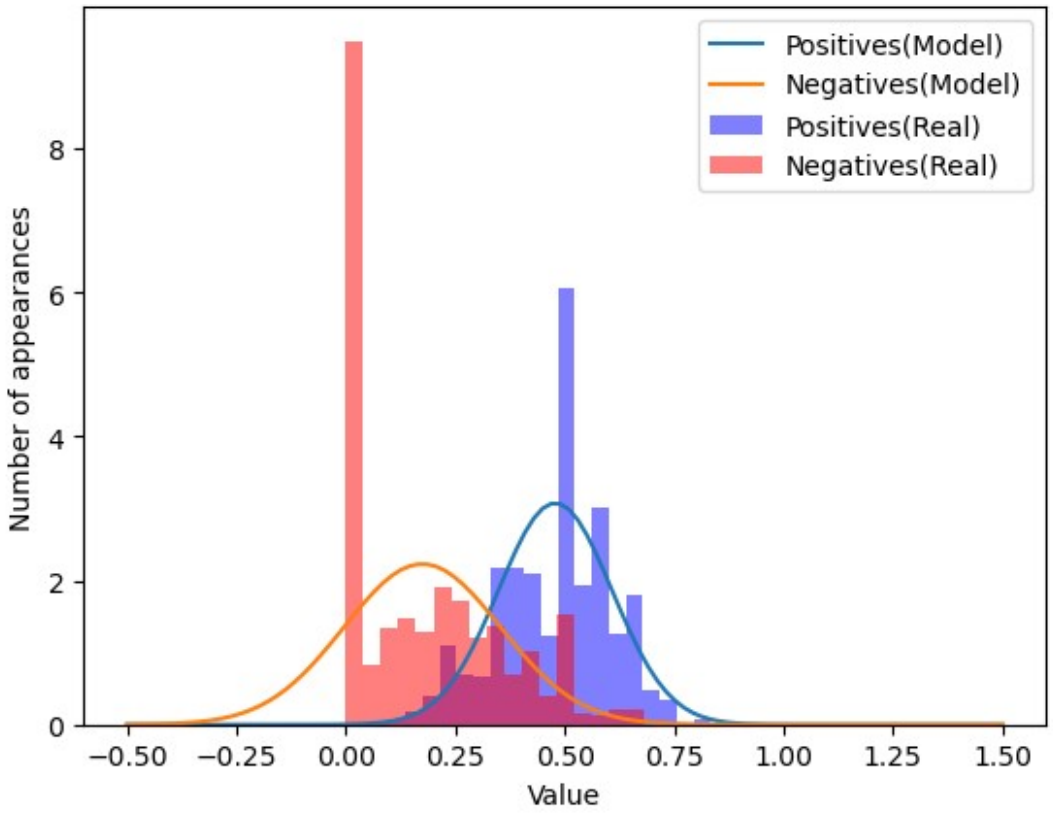
```
Pos mean: 0.48 std: 0.13
Neg mean: 0.17 std: 0.18
```

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)

```
normalisedPos = scipy.stats.norm(mu_pos, std_pos)
normalisedNeg = scipy.stats.norm(mu_neg, std_neg)

space = np.linspace(-0.5, 1.5, 100)
plt.plot(space, normalisedPos.pdf(space), label='Positives(Model)')
plt.plot(space, normalisedNeg.pdf(space), label='Negatives(Model)')
plt.hist(train_dataset.pos[:, FOI], bins=20, alpha=0.5,
label='Positives(Real)', color='b', density=True)
plt.hist(train_dataset.neg[:, FOI], bins=20,  alpha=0.5,
label='Negatives(Real)', color='r', density=True)
plt.legend()
plt.xlabel('Value')
plt.ylabel('Number of appearances')
plt.show()
```



Naimplementujte binární generativní klasifikátor. Při konstrukci přijímá dvě rozložení poskytující metodu `.pdf()` a odpovídající apriorní pravděpodobnost tříd. Dbejte, aby Vám uživatel nemohl zadat neplatné apriorní pravděpodobnosti. Jako všechny klasifikátory v tomto projektu poskytuje metodu `prob_class_1()`.

(9 řádků)

```
class GenerativeClassifier2Class:
    def __init__(self, pos, neg, prob1: float, prob2: float):
        self.__pos = pos
        self.__neg = neg
        self.__prob1 = prob1
        self.__prob2 = prob2
        if self.__prob1 + self.__prob2 != 1:
            raise ValueError('Probabilities must sum to 1')

    def prob_class_1(self, xs):
        posPdf = self.__pos.pdf(xs)
        negPdf = self.__neg.pdf(xs)
        posProduct = posPdf * self.__prob1
        negProduct = negPdf * self.__prob2
        sum = posProduct + negProduct
        return posProduct / sum
```

Nainstancujte dva generativní klasifikátory: jeden s rovnoměrnými priory a jeden s apriorní pravděpodobností 0.75 pro třídu 0 (negativní příklady). Pomocí funkce `evaluate()` vyhodnotíte jejich úspěšnost na validačních datech.

(2 řádky)

```
classifier_flat_prior = GenerativeClassifier2Class(normalisedPos,
normalisedNeg, 0.5, 0.5)
classifier_full_prior = GenerativeClassifier2Class(normalisedPos,
normalisedNeg, 0.25, 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
```

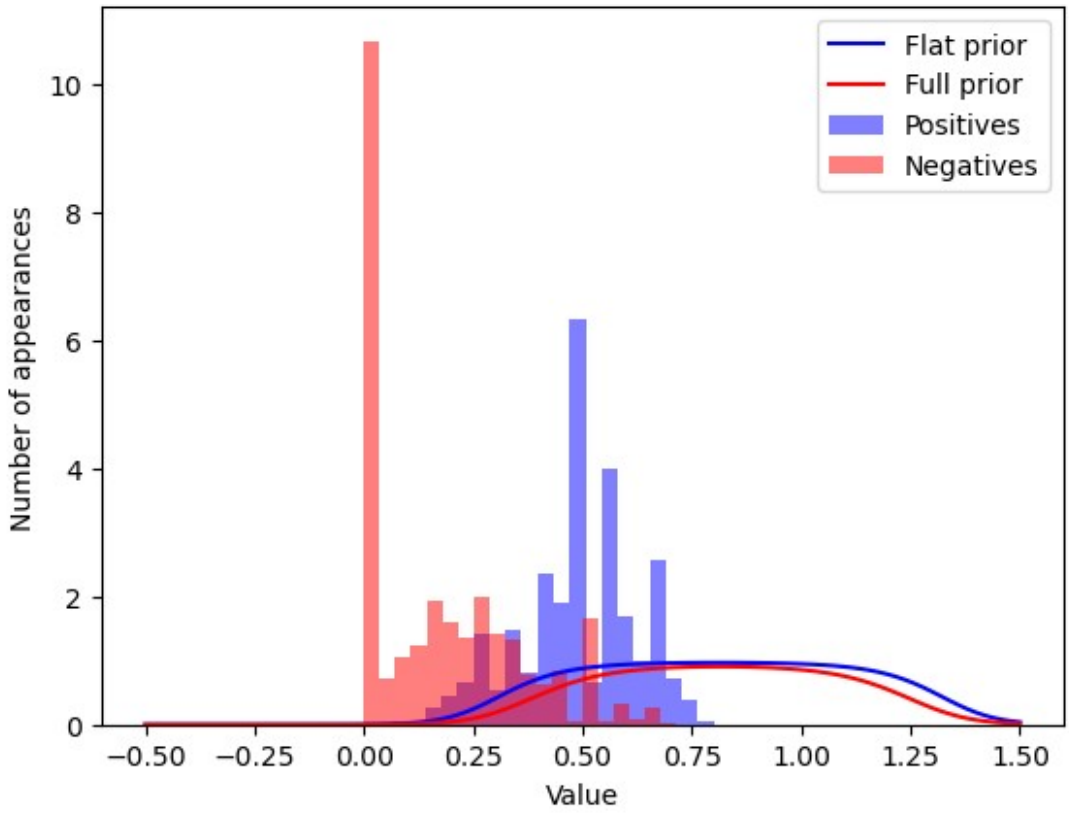
Vykreslete průběh posteriorní pravděpodobnosti třídy 1 jako funkci příznaku 5, opět v rozsahu <-0.5; 1.5> pro oba klasifikátory. Do grafu zakreslete i histogramy rozložení trénovacích dat, opět s `density=True` pro zachování dynamického rozsahu.

(8 řádků)

```
xs = np.linspace(-0.5, 1.5, 100)

plt.plot(xs, classifier_flat_prior.prob_class_1(xs), label='Flat
prior', color='b')
plt.plot(xs, classifier_full_prior.prob_class_1(xs), label='Full
prior', color='r')

plt.hist(val_dataset.pos[:, FOI], bins=20, alpha=0.5,
label='Positives', color='b', density=True)
plt.hist(val_dataset.neg[:, FOI], bins=20, alpha=0.5,
label='Negatives', color='r', density=True)
plt.legend()
plt.xlabel('Value')
plt.ylabel('Number of appearances')
plt.show()
```



Diskriminativní klasifikátory

V následující části budete pomocí (lineární) logistické regrese přímo modelovat posteriorní pravděpodobnost třídy 1. Modely budou založeny čistě na NumPy, takže nemusíte instalovat nic dalšího. Nabitějších toolkitů se dočkáte ve třetím projektu.

```
def logistic_sigmoid(x):
    return np.exp(-np.logaddexp(0, -x))

def binary_cross_entropy(probs, targets):
    return np.sum(-targets * np.log(probs) - (1-targets)*np.log(1-
probs))

class LogisticRegressionNumpy:
    def __init__(self, dim):
        self.w = np.array([0.0] * dim)
        self.b = np.array([0.0])

    def prob_class_1(self, x):
        return logistic_sigmoid(x @ self.w + self.b)
```

Diskriminativní klasifikátor očekává, že dostane vstup ve tvaru `[N, F]`. Pro práci na jediném příznaku bude tedy zapotřebí vyřezávat příslušná data v správném formátu (`[N, 1]`).

Doimplementujte třídu `FeatureCutter` tak, aby to zařizovalo volání její instance. Který příznak se použije, nechť je konfigurováno při konstrukci.

Může se Vám hodit `np.newaxis`.

(2 řádky)

```
class FeatureCutter:
    def __init__(self, fea_id):
        self.fea_id = fea_id

    def __call__(self, x):
        return x[:, self.fea_id][:, np.newaxis]
```

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. Neuvažujte žádné minibatche, aktualizujte váhy vždy na celém trénovacím datasetu. Po každém kroku vyhodnoťte model na validačních datech. Jako model vracejte ten, který dosáhne nejlepší validační přesnosti. Jako loss použijte binární cross-entropii a logujte průměr na vzorek. Pro výpočet validační přesnosti využijte funkci `evaluate()`. Oba průběhy vracejte jako obyčejné seznamy.

(cca 11 řádků)

```
def train_logistic_regression(nb_epochs, lr, in_dim,
fea_preprocessor):
    model = LogisticRegressionNumpy(in_dim)
    best_model = copy.deepcopy(model)
    losses = []
    accuracies = []

    train_X = fea_preprocessor(train_dataset.xs)
    train_t = train_dataset.targets

    val_X = fea_preprocessor(val_dataset.xs)
    val_t = val_dataset.targets

    for n in range(nb_epochs):
        y = model.prob_class_1(train_X)
        loss = binary_cross_entropy(y, train_t)
        losses.append(loss / len(train_t))

        # Gradient descent step
        error = y - train_t
        grad_w = np.dot(train_X.T, error) / len(train_t)
        grad_b = np.sum(error) / len(train_t)
        model.w -= lr * grad_w
        model.b -= lr * grad_b

        # Evaluate on validation data
```

```
        accuracy = evaluate(model, val_X, val_t)
        accuracies.append(accuracy)

        if len(accuracies) > 1 and accuracy > max(accuracies[:-1]):
            best_model = copy.deepcopy(model)

    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.

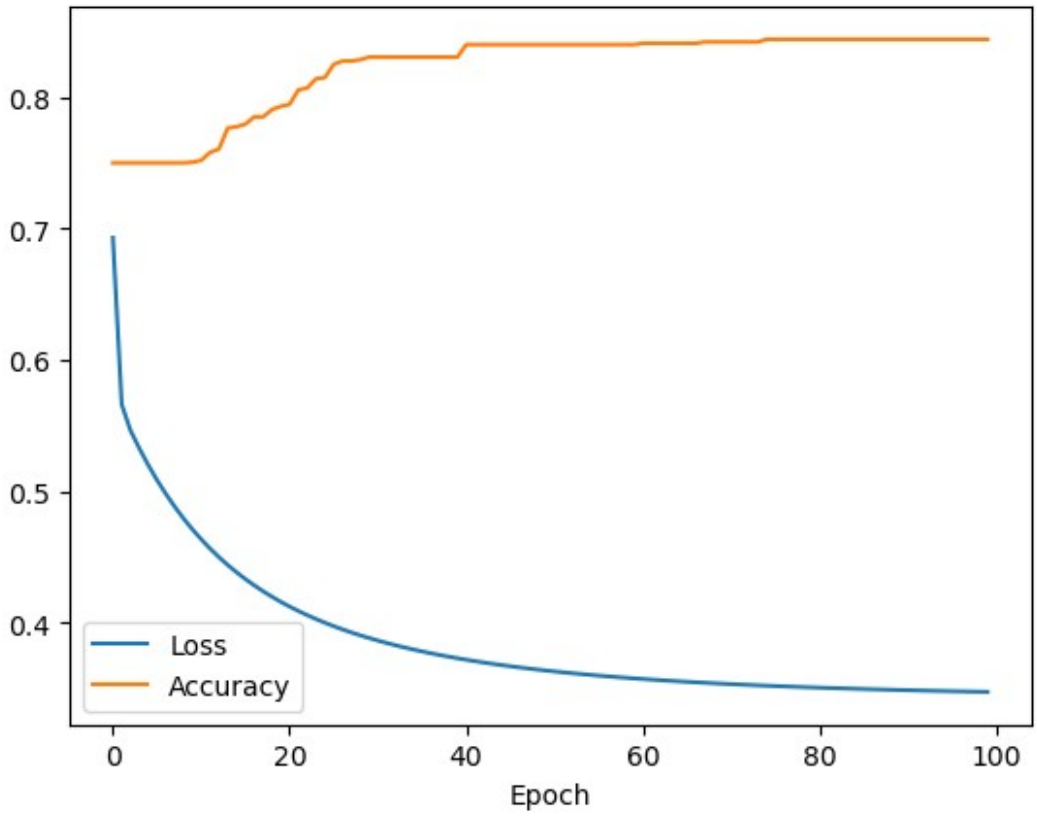
V druhém grafu vykreslete histogramy trénovacích dat a pravděpodobnost třídy 1 pro x od -0.5 do 1.5, podobně jako výše u generativních klasifikátorů.

(1 + 5 + 8 řádků)

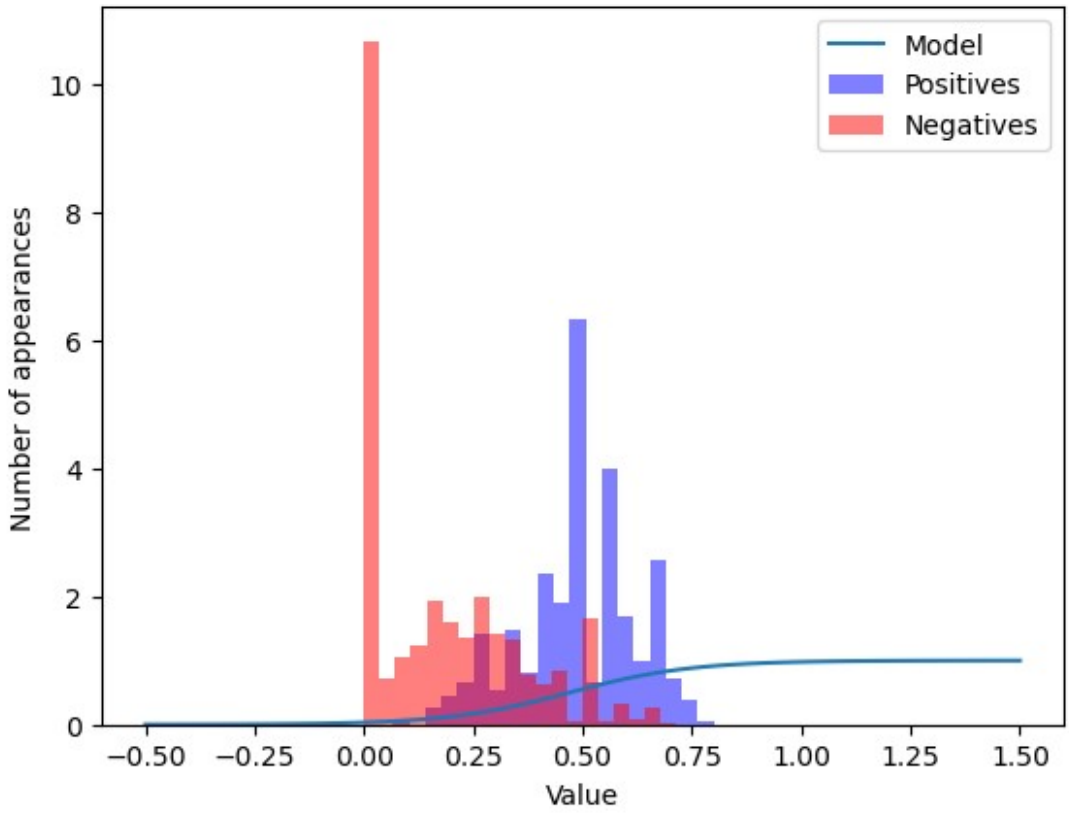
```
disc_fea5, losses, accuracies = train_logistic_regression(100, 5, 1,
FeatureCutter(FOI))

plt.plot(losses, label='Loss')
plt.plot(accuracies, label='Accuracy')
plt.legend()
plt.xlabel('Epoch')
plt.show()
print('w', disc_fea5.w.item(), 'b', disc_fea5.b.item())

xs = np.linspace(-0.5, 1.5, 100)
plt.plot(xs, logistic_sigmoid(xs * disc_fea5.w + disc_fea5.b),
label='Model')
plt.hist(val_dataset.pos[:, FOI], bins=20, alpha=0.5,
label='Positives', color='b', density=True)
plt.hist(val_dataset.neg[:, FOI], bins=20, alpha=0.5,
label='Negatives', color='r', density=True)
plt.legend()
plt.xlabel('Value')
plt.ylabel('Number of appearances')
plt.show()
print('disc_fea5:', evaluate(disc_fea5, val_dataset.xs[:, FOI][:,
np.newaxis], val_dataset.targets))
```




```
w 7.197054833406055 b -3.385253333656651
```




```
disc_fea5: 0.844
```

Všechny vstupní příznaky

V posledním cvičení natrénujete logistickou regresi, která využije všechn sedm vstupních příznaků. Zavolejte funkci z předchozího cvičení, opět vykreslete průběh trénovací loss a validační přesnosti. Měli byste se dostat nad 90 % přesnosti.

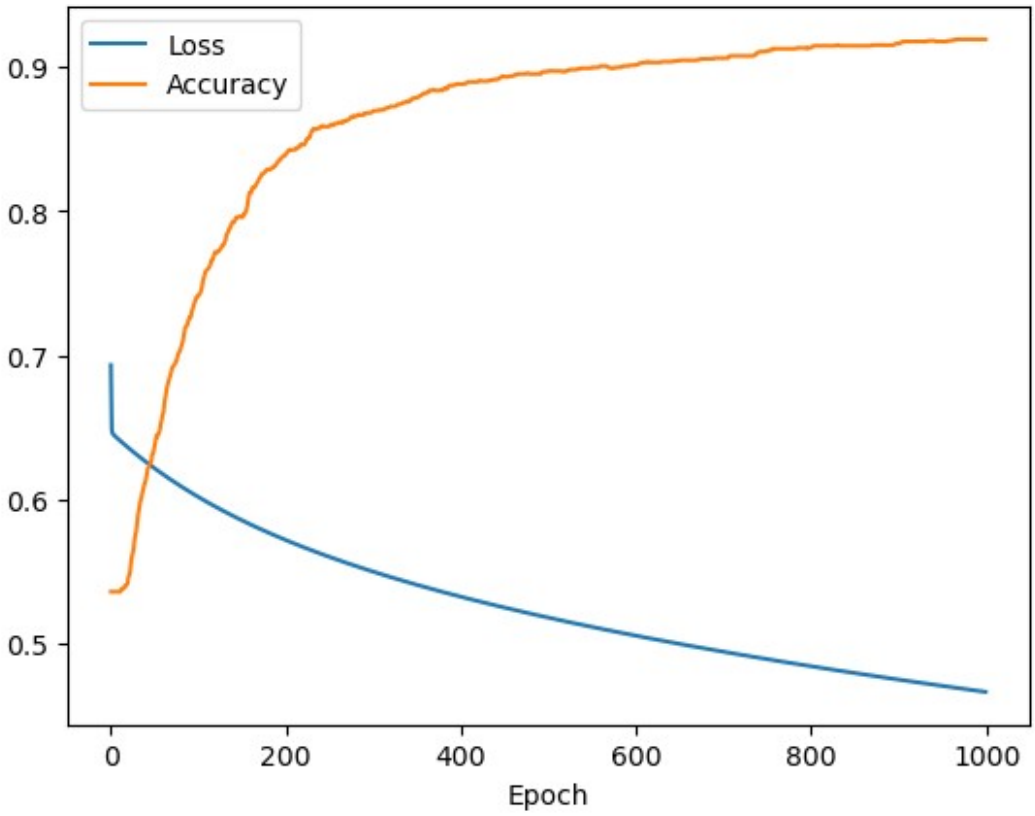
Může se Vám hodit `lambda` funkce.

(1 + 5 řádků)

```
disc_full_fea, losses, accuracies = train_logistic_regression(1000,
0.001, train_dataset.xs.shape[1], lambda x: x)

plt.plot(losses, label='Loss')
plt.plot(accuracies, label='Accuracy')
plt.legend()
plt.xlabel('Epoch')
plt.show()

print('disc_full_fea:', evaluate(disc_full_fea, val_dataset.xs,
val_dataset.targets))
```




```
disc_full_fea: 0.919
```

Závěrem

Konečně vyhodnoťte všech pět vytvořených klasifikátorů na testovacích datech. Stačí doplnit jejich názvy a předat jim odpovídající příznaky. Nezapomeňte, že u logistické regrese musíte zopakovat formátovací krok z `FeatureCutter`u.

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

print('Baseline:', evaluate(baseline, xs_full, 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(disc_fea5,
xs_foi[:,np.newaxis], targets))
print('logistic regression all features:', evaluate(disc_full_fea,
xs_full, targets))
```

```
Baseline: 0.75
Generative classifier (w/o prior): 0.8
Generative classifier (correct): 0.847
Logistic regression: 0.853
logistic regression all features: 0.9155
```

Blahopřejeme ke zvládnutí projektu! Nezapomeňte (1) spustit celý notebook načisto (Kernel -> Restart & Run all), (2) zkontrolovat, že všechny výpočty prošly podle očekávání, a (3) před odevzdáním pojmenovat soubory loginem vedoucího týmu.

Mimochodem, vstupní data nejsou synteticky generovaná. Nasbírali jsme je z baseline řešení historicky prvního SUI projektu; vaše klasifikátory v tomto projektu predikují, že daný hráč vyhraje dicewars, takže by se daly použít jako heuristika pro ohodnocování listových uzlů ve stavovém prostoru hry. Pro představu, data jsou 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. Nejeden projekt v ročníku 2020 realizoval požadované "strojové učení" v agentovi hrajicím dicewars kopií domácí úlohy, která byla předchůdkyní tohoto projektu.