Vítejte u druhého 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 kříži 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.

In [ ]:
```python
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)

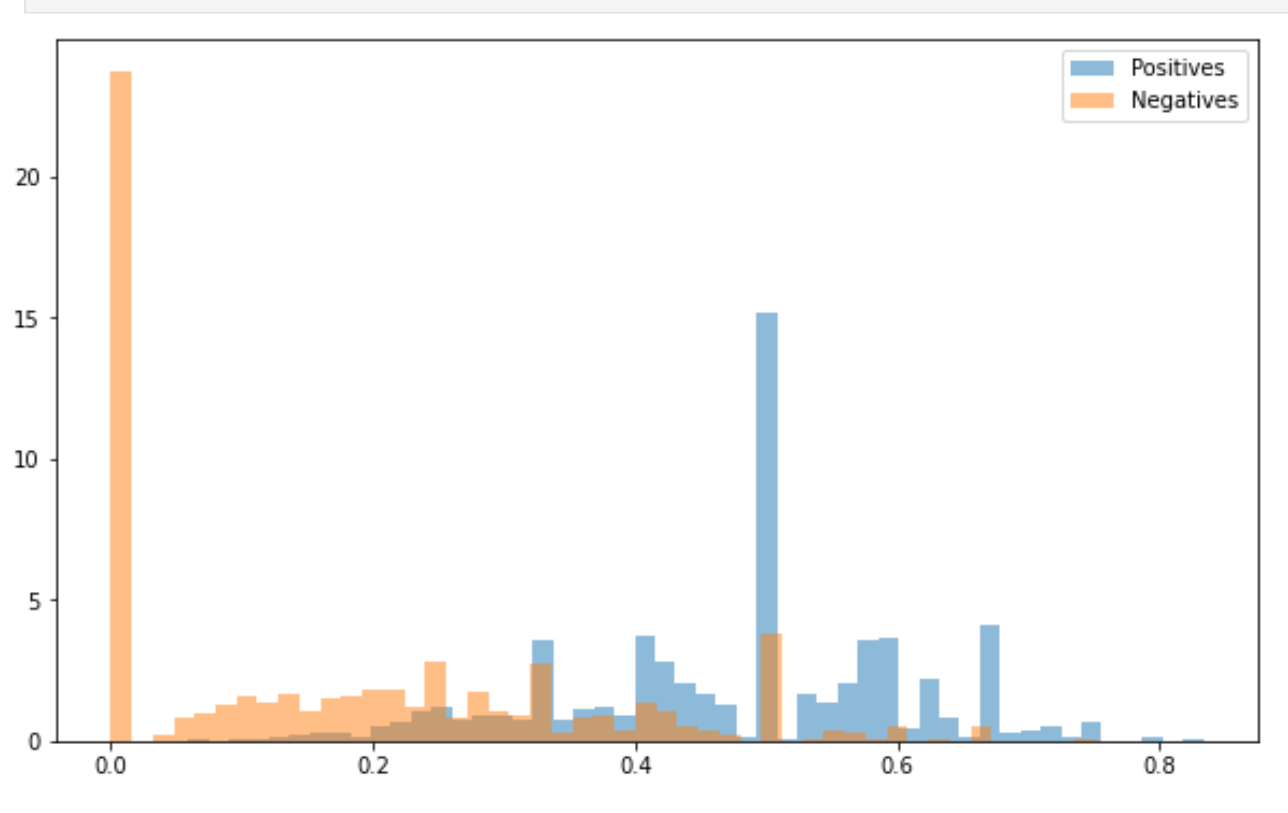
In [ ]:
```python
class BinaryDataset:
    def __init__(self, positives, negatives):
        self.pos = np.loadtxt(positives)
        self.neg = np.loadtxt(negatives)
        self.xs = np.vstack((self.pos, self.neg))
        self.targets = np.concatenate((np.repeat(1, self.pos.shape[0]), np.repeat(0, 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)
```

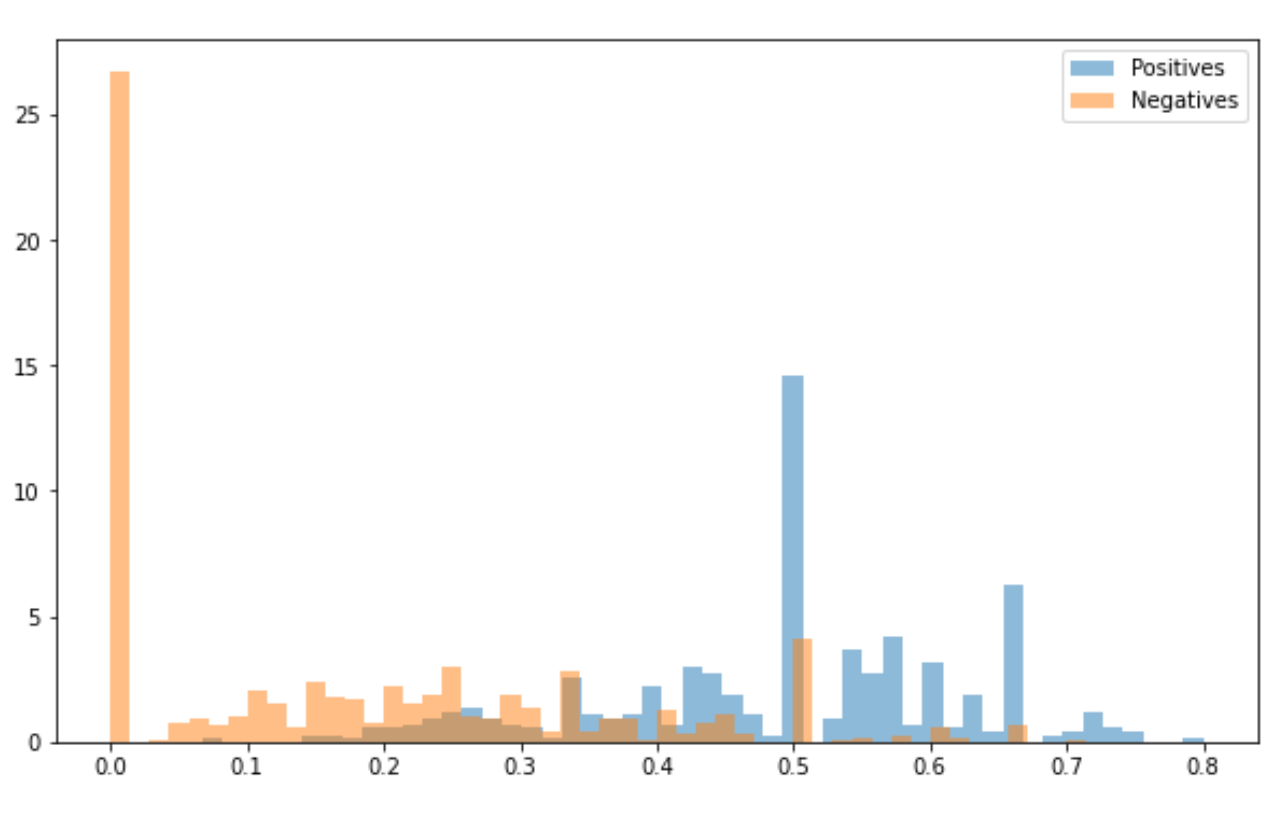
```
positives (2280, 7)
negatives (6841, 7)
xs (9121, 7)
targets (9121,)
```

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)

In [ ]:
```python
FOI = 5  # Feature Of Interest

def plot_data(poss, negs):
    plt.figure(figsize=(10, 6))
    plt.hist(poss, alpha=0.5, label='Positives', bins=50, density=True)
    plt.hist(negs, alpha=0.5, label='Negatives', bins=50, density=True)
    plt.legend()
    plt.show()

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





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)

In [ ]:
```python
def evaluate(classifier, inputs, targets):
    posterior = classifier.prob_class_1(inputs) > 0.5
    accuracy = (posterior == targets)
    return np.sum(accuracy)/len(accuracy)

class Dummy: # zkusebni klasifikator
    def prob_class_1(self, xs): # vraci pole posteriornich pst
        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)

In [ ]:
```python
class PriorClassifier:
    def __init__(self, class_tip):
        self.class_tip = class_tip

    def prob_class_1(self, data):
        return np.full(data.shape[0], self.class_tip) # vraci pole stejneho tvaru jako data, kde vsechny hodnoty jsou self.prediction_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
```

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)

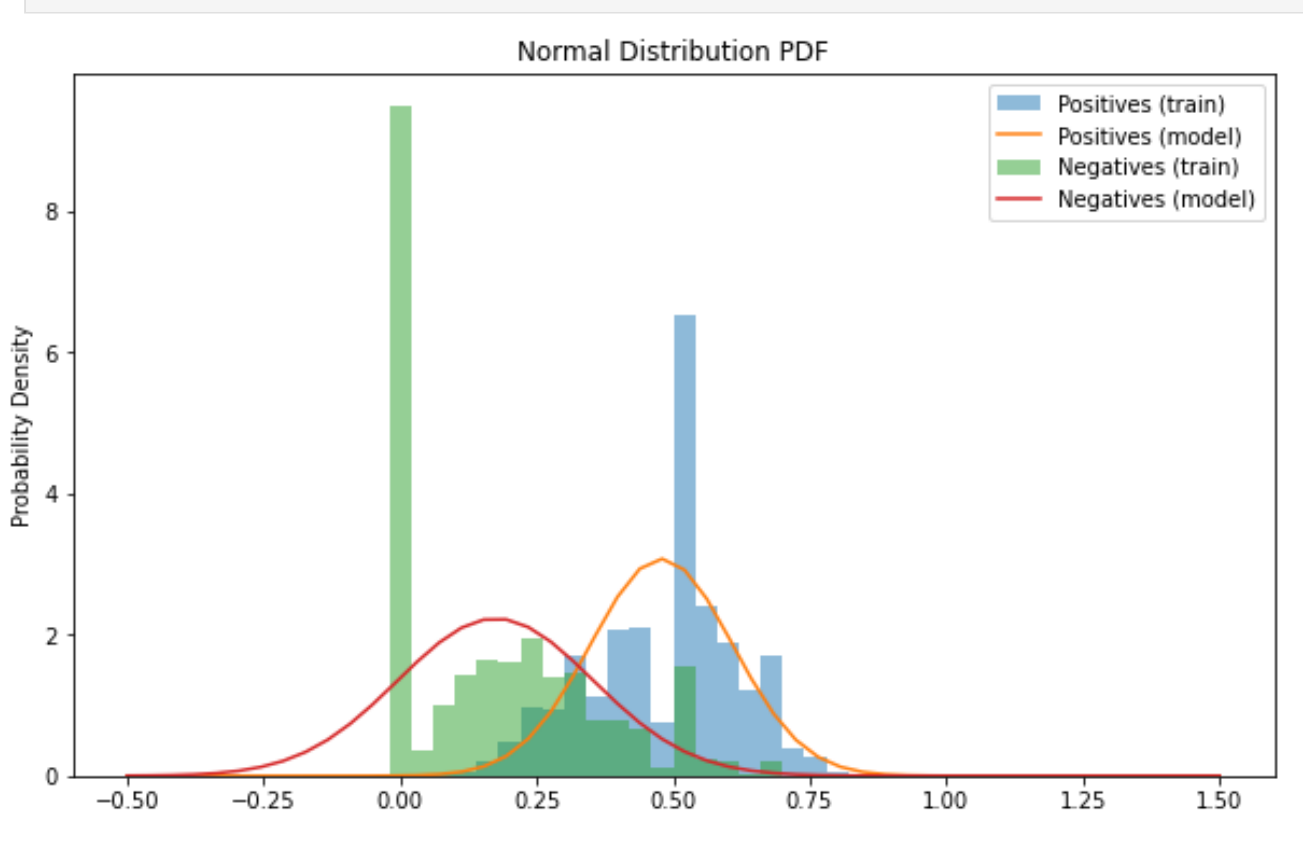
In [ ]:
```python
def mle_gauss_1d(data):
    return np.mean(data), np.sqrt(np.mean((data - np.mean(data))**2))
        # u = (1/n)*sum(x_i),  sigma = sqrt((1/n)*sum(x_i-u)^2)

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)

In [ ]:
```python
x_size = np.linspace(-0.5, 1.5)

plt.figure(figsize=(10, 6))

pdf_pos = scipy.stats.norm.pdf(x_size, loc=mu_pos, scale=std_pos)
pdf_neg = scipy.stats.norm.pdf(x_size, loc=mu_neg, scale=std_neg)

plt.hist(train_dataset.pos[:, FOI], range = [-0.5, 1.5], alpha=0.5, label='Positives (train)', bins=50, density=True)
plt.plot(x_size, pdf_pos, label='Positives (model)')

plt.hist(train_dataset.neg[:, FOI],  range = [-0.5, 1.5],alpha=0.5, label='Negatives (train)', bins=50, density=True)
plt.plot(x_size, pdf_neg, label='Negatives (model)')

plt.ylabel('Probability Density')
plt.title('Normal Distribution PDF')
plt.legend()
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)

In [ ]:
```python
class GenerativeClassifier2Class:
    def __init__(self, dist_pos, dist_neg, apriori_prob):
        assert(np.sum(apriori_prob) == 1)
        self.dist_pos = dist_pos
        self.dist_neg = dist_neg
        self.apriori_prob_neg = apriori_prob[0]
        self.apriori_prob_pos = apriori_prob[1]

    def prob_class_1(self, data):
        denominator = (self.dist_pos.pdf(data)*self.apriori_prob_pos) + (self.dist_neg.pdf(data)*self.apriori_prob_neg)
        class_pos = (self.dist_pos.pdf(data)*self.apriori_prob_pos)/denominator
        return class_pos
```

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)

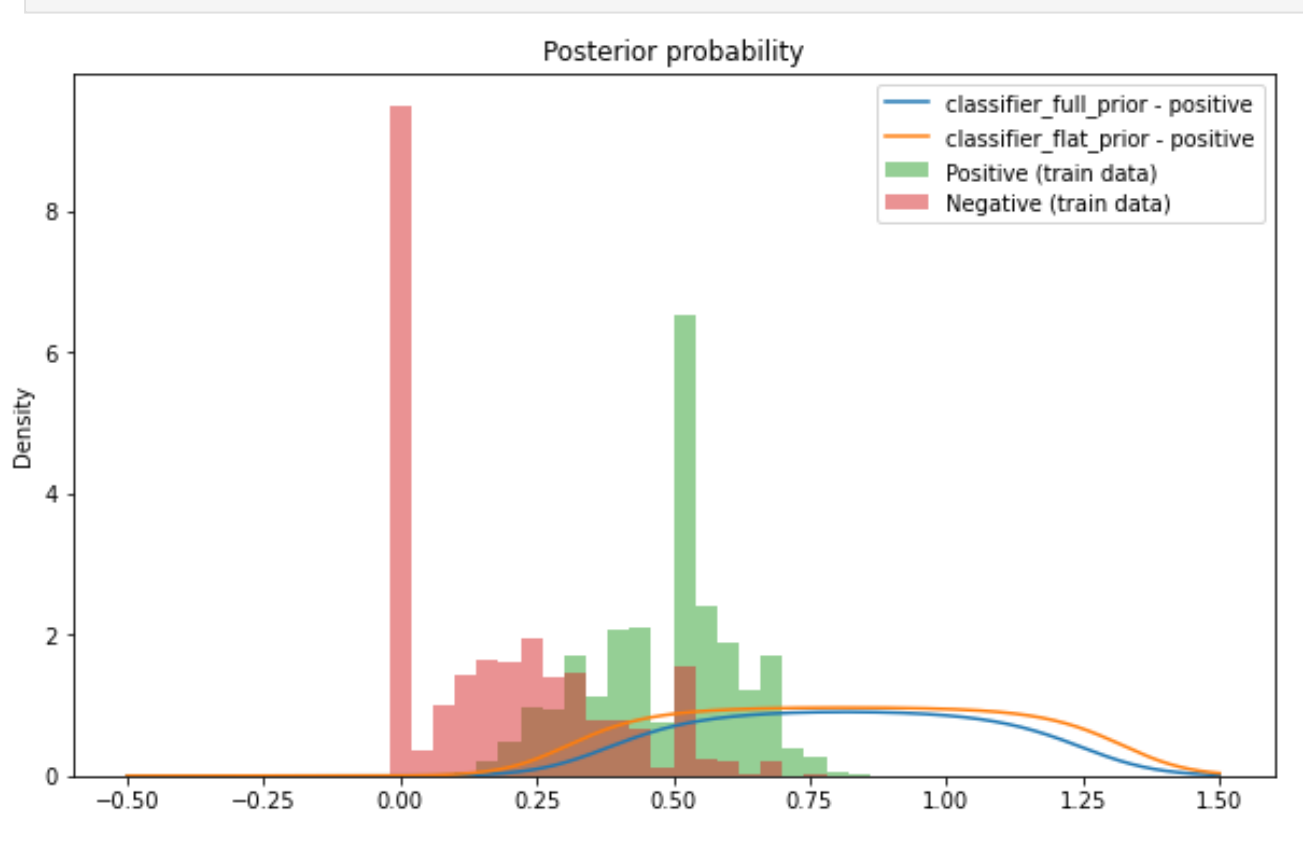
In [ ]:
```python
classifier_flat_prior = GenerativeClassifier2Class(scipy.stats.norm(loc=mu_pos, scale=std_pos), scipy.stats.norm(loc=mu_neg, scale=std_neg), [0.5, 0.5])
classifier_full_prior = GenerativeClassifier2Class(scipy.stats.norm(loc=mu_pos, scale=std_pos), scipy.stats.norm(loc=mu_neg, scale=std_neg), [0.75, 0.25])

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)

In [ ]:
```python
x_size = np.linspace(-0.5, 1.5,2000)

plt.figure(figsize=(10, 6))

plt.plot(x_size, classifier_full_prior.prob_class_1(x_size), label='classifier_full_prior - positive')
plt.plot(x_size, classifier_flat_prior.prob_class_1(x_size), label='classifier_flat_prior - positive')

plt.hist(train_dataset.pos[:, FOI], range = [-0.5, 1.5], alpha=0.5, label='Positive (train data)', bins=50, density=True)
plt.hist(train_dataset.neg[:, FOI], range = [-0.5, 1.5], alpha=0.5, label='Negative (train data)', bins=50, density=True)

plt.ylabel('Density')
plt.title('Posterior probability')
plt.legend()
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.

In [ ]:
```python
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)  # w1
        self.b = np.array([0.0])        # w0

    def prob_class_1(self, x): # tohle dato x je s danou pst. ve tride 1
        return logistic_sigmoid(x @ self.w + self.b) # w1*x + w0
```

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)

In [ ]:
```python
class FeatureCutter:
    def __init__(self, fea_id):
        self.fea_id = fea_id

    def __call__(self, x):
        return x[:, self.fea_id][:, np.newaxis] # explicitne rikame, ze v druhe ose mame rozmer 1
```

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)

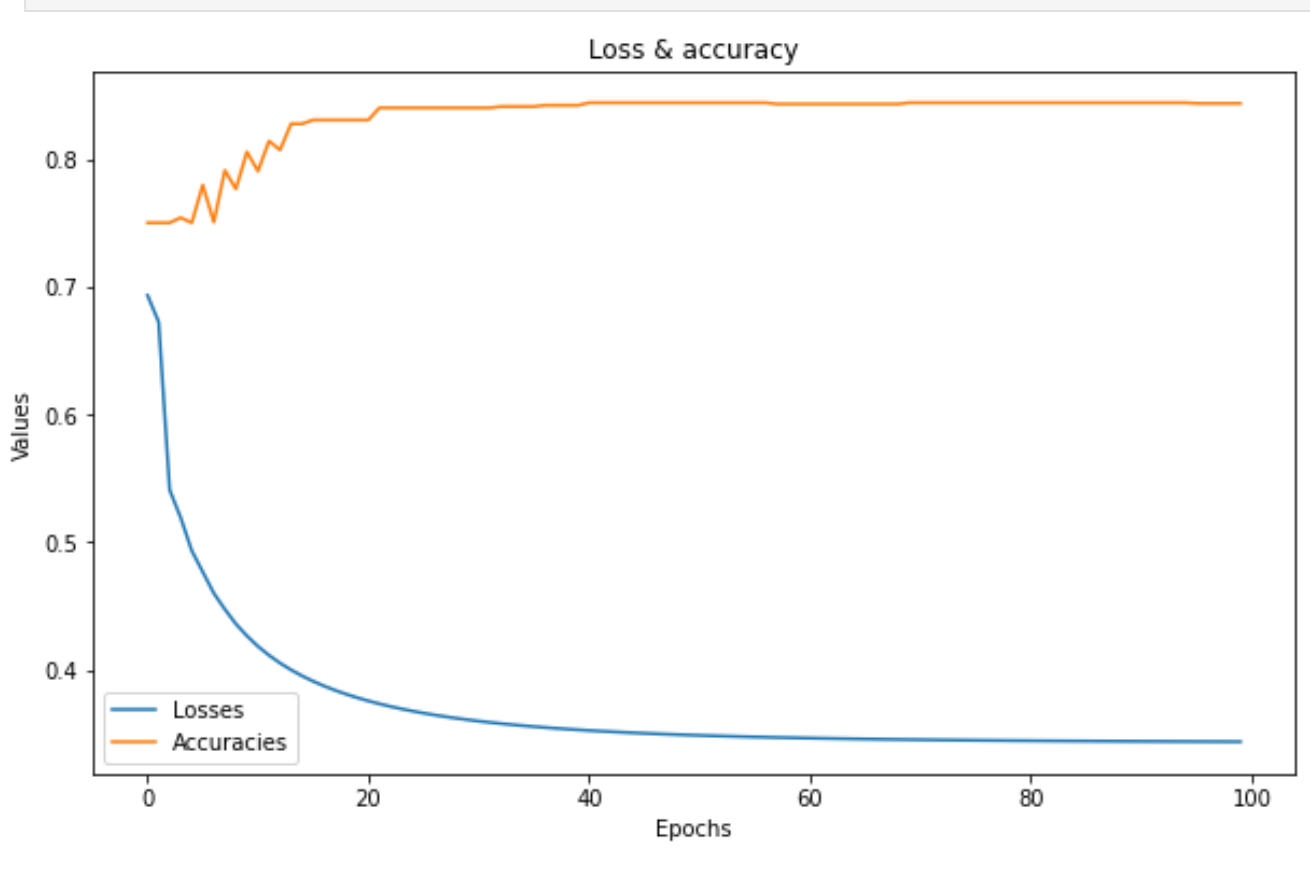
In [ ]:
```python
def train_logistic_regression(nb_epochs, lr, in_dim, fea_preprocessor):
    model = LogisticRegressionNumpy(in_dim)
    best_model = copy.deepcopy(model)
    losses = []
    accuracies = []

    train_X = fea_preprocessor(train_dataset.xs)
    train_t = train_dataset.targets
    val_t = val_dataset.targets
    best_accuracy = 0

    for _ in range(nb_epochs): # budeme trenovat po nb_epochs pocet epoch
        # pst, ze je to trida 1 pro dana data (na zacatku mame w0, w1 = 0, tj. sigmoida v 0 ma hodnotu 0.5, tedy ze nevi, do jake tridy zaradit)
        data_prob1 = model.prob_class_1(train_X)
        # update parametru
        grad = (data_prob1 - train_t).dot(train_X)
        model.b = model.b - lr*np.sum(data_prob1 - train_t)
        model.w = model.w - lr*grad
        # presnost
        accuracy = evaluate(model, fea_preprocessor(val_dataset.xs), val_t)
        accuracies.append(accuracy)
        # chybovost
        loss = binary_cross_entropy(data_prob1, train_t)/(len(train_X))
        losses.append(loss)

        if best_accuracy < accuracy:
            best_accuracy = accuracy
            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.

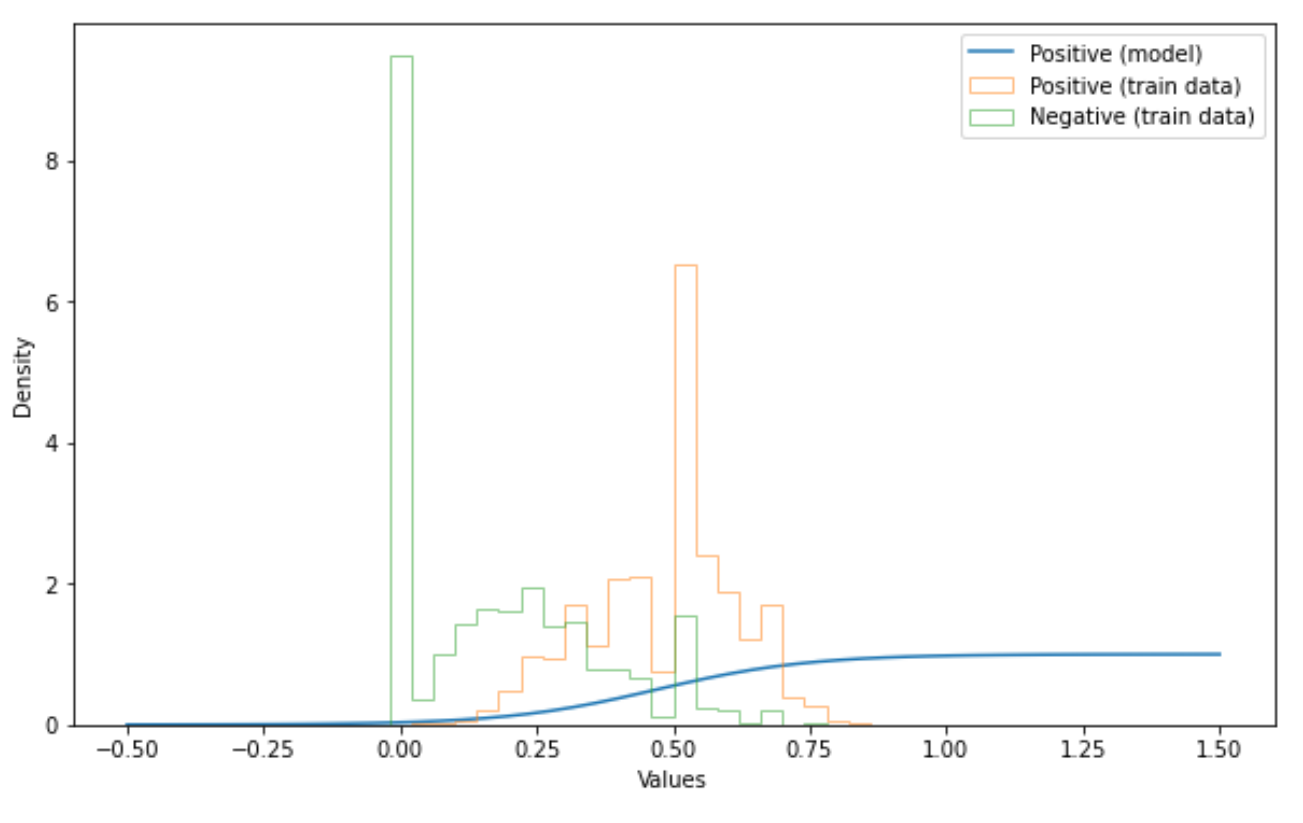
V druhém grafu vykreslete histogramy trénovacích dat a pravděpodobnost třídy 1 pro x od -0.5 do 1.5, podobně jako výše u generativních klasifikátorů.

(1 + 5 + 8 řádků)

In [ ]:
```python
disc_fea5, losses, accuracies = train_logistic_regression(100, 0.001, 1, FeatureCutter(FOI))
##############################
plt.figure(figsize=(10, 6))
plt.plot(losses, label='Losses')
plt.plot(accuracies, label='Accuracies')
plt.ylabel('Values')
plt.xlabel('Epochs')
plt.title('Loss & accuracy')
plt.legend()
plt.show()
##############################
print('w', disc_fea5.w.item(), 'b', disc_fea5.b.item())
##############################
x_size = np.linspace(-0.5, 1.5)
plt.figure(figsize=(10, 6))
plt.plot(x_size, disc_fea5.prob_class_1(x_size[:, np.newaxis]), label='Positive (model)')
plt.hist(train_dataset.pos[:, FOI], range = [-0.5, 1.5], alpha=0.5, histtype='step', label='Positive (train data)', bins=50, density=True)
plt.hist(train_dataset.neg[:, FOI], range = [-0.5, 1.5], alpha=0.5, histtype='step', label='Negative (train data)', bins=50, density=True)
plt.ylabel('Density')
plt.xlabel('Values')
plt.legend()
plt.show()
##############################
print('disc_fea5 - accuracy:', evaluate(disc_fea5, val_dataset.xs[:, FOI][:, np.newaxis], val_dataset.targets))
```



```
w 7.222447064152967 b -3.3949621316355643
```



```
disc_fea5 - accuracy: 0.844
```

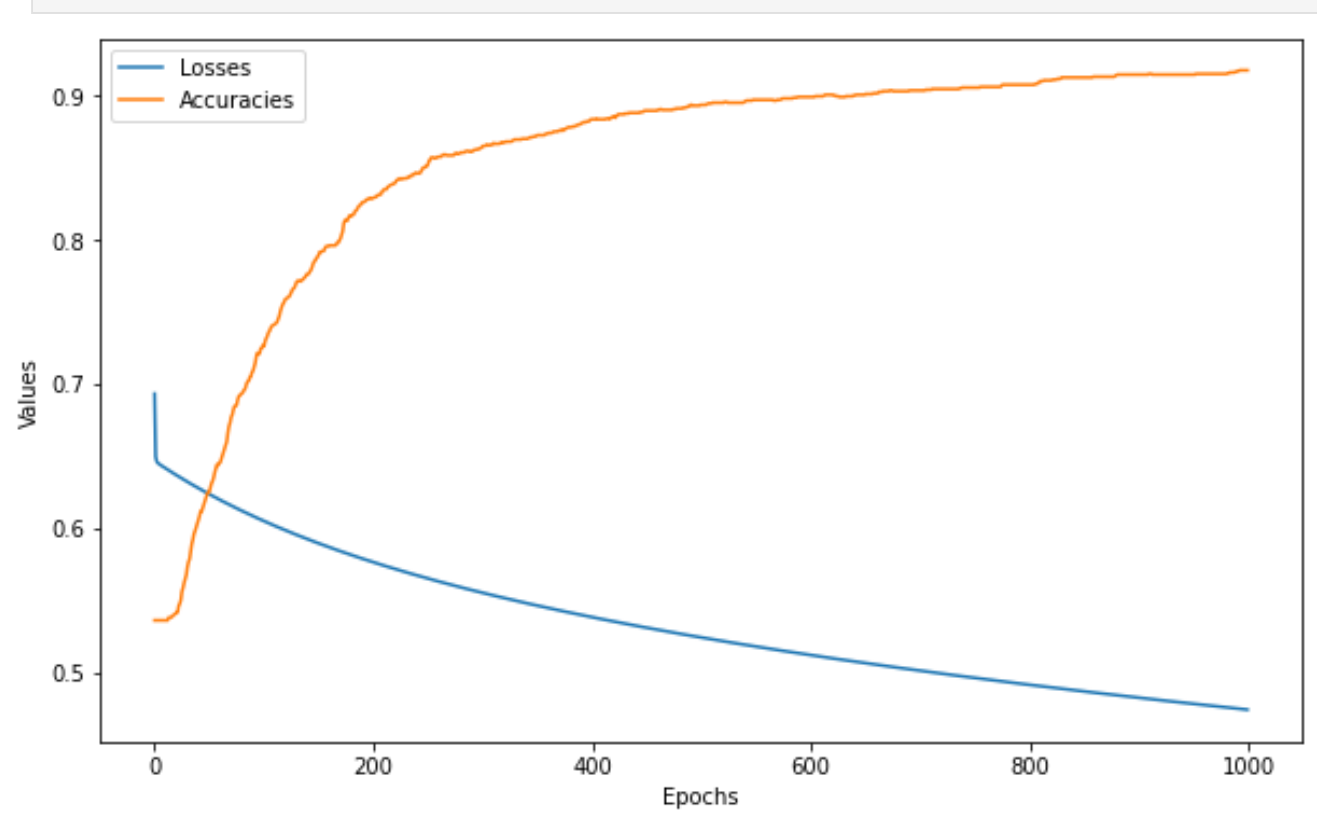
Všechny vstupní příznaky

V posledním cvičení natrénujete logistickou regresi, která využije všech 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)

In [ ]:
```python
disc_full_fea, losses, accuracies = train_logistic_regression(1000, 0.0000001, 7, lambda x:x)

plt.figure(figsize=(10, 6))
plt.plot(losses, label='Losses')
plt.plot(accuracies, label='Accuracies')
plt.ylabel('Values')
plt.xlabel('Epochs')
plt.legend()
plt.show()

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



```
disc_full_fea - accuracy: 0.9175
```

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.

In [ ]:
```python
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: 0.9135
```

Blahopřejeme ke zvládnutí projektu! Nezapomeňte spustit celý notebook načisto (Kernel -> Restart & Run all) a zkontrolovat, že všechny výpočty prošly podle očekávání.

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" kopií domácí úlohy.