SU-2019-LAB2-0036501052

November 3, 2019

Sveuilite u Zagrebu
Fakultet elektrotehnike i raunarstva

0.1 Strojno uenje 2019/2020

http://www.fer.unizg.hr/predmet/su

0.1.1 Laboratorijska vjeba 2: Linearni diskriminativni modeli

Verzija: 1.3
Zadnji put aurirano: 27. rujna 2019.

Objavljeno: **30. rujna 2019.**
Rok za predaju: **4. studenog 2019. u 07:00h**

0.1.2 Upute

Prva laboratorijska vjeba sastoji se od est zadataka. U nastavku slijedite upute navedene u elijama s tekstom. Rjeavanje vjebe svodi se na **dopunjavanje ove biljenice**: umetanja elije ili vie njih **ispod** teksta zadatka, pisanja odgovarajueg kôda te evaluiranja elija.

Osigurajte da u potpunosti **razumijete** kôd koji ste napisali. Kod predaje vjebe, morate biti u stanju na zahtjev asistenta (ili demonstratora) preinaiti i ponovno evaluirati Va kôd. Nadalje, morate razumjeti teorijske osnove onoga to radite, u okvirima onoga to smo obradili na predavanju. Ispod nekih zadataka moete nai i pitanja koja slue kao smjernice za bolje razumijevanje gradiva (**nemojte pisati** odgovore na pitanja u biljenicu). Stoga se nemojte ograniiti samo na to da rijeite zadatak, nego slobodno eksperimentirajte. To upravo i jest svrha ovih vjebi.

Vjebe trebate raditi **samostalno**. Moete se konzultirati s drugima o naelnom nainu rjeavanja, ali u konanici morate sami odraditi vjebu. U protivnome vjeba nema smisla.

[1]:
```
# Uitaj osnovne biblioteke...
import numpy as np
import sklearn
import mlutils
import matplotlib.pyplot as plt
```

```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

0.2 Zadatci

0.2.1 1. Linearna regresija kao klasifikator

U prvoj laboratorijskoj vjebi koristili smo model linearne regresije za, naravno, regresiju. Meutim, model linearne regresije moe se koristiti i za **klasifikaciju**. Iako zvui pomalo kontraintuitivno, zapravo je dosta jednostavno. Naime, cilj je nauiti funkciju $f(\mathbf{x})$ koja za negativne primjere predvia vrijednost 1, dok za pozitivne primjere predvia vrijednost 0. U tom sluaju, funkcija $f(\mathbf{x}) = 0.5$ predstavlja granicu izmeu klasa, tj. primjeri za koje vrijedi $h(\mathbf{x}) \geq 0.5$ klasificiraju se kao pozitivni, dok se ostali klasificiraju kao negativni.

Klasifikacija pomou linearne regresije implementirana je u razredu `RidgeClassifier`. U sljedeim podzadatcima **istrenirajte** taj model na danim podatcima i **prikaite** dobivenu granicu izmeu klasa. Pritom iskljuite regularizaciju ($\alpha = 0$, odnosno `alpha=0`). Takoer i ispiite **tonost** vaeg klasifikacijskog modela (smijete koristiti funkciju `metrics.accuracy_score`). Skupove podataka vizualizirajte koritenjem pomone funkcije `plot_clf_problem(X, y, h=None)` koja je dostupna u pomonom paketu `mlutils` (datoteku `mlutils.py` moete preuzeti sa stranice kolegija). `X` i `y` predstavljaju ulazne primjere i oznake, dok `h` predstavlja funkciju predikcije modela (npr. `model.predict`).

U ovom zadatku cilj je razmotriti kako se klasifikacijski model linearne regresije ponaa na linearno odvojim i neodvojivim podatcima.

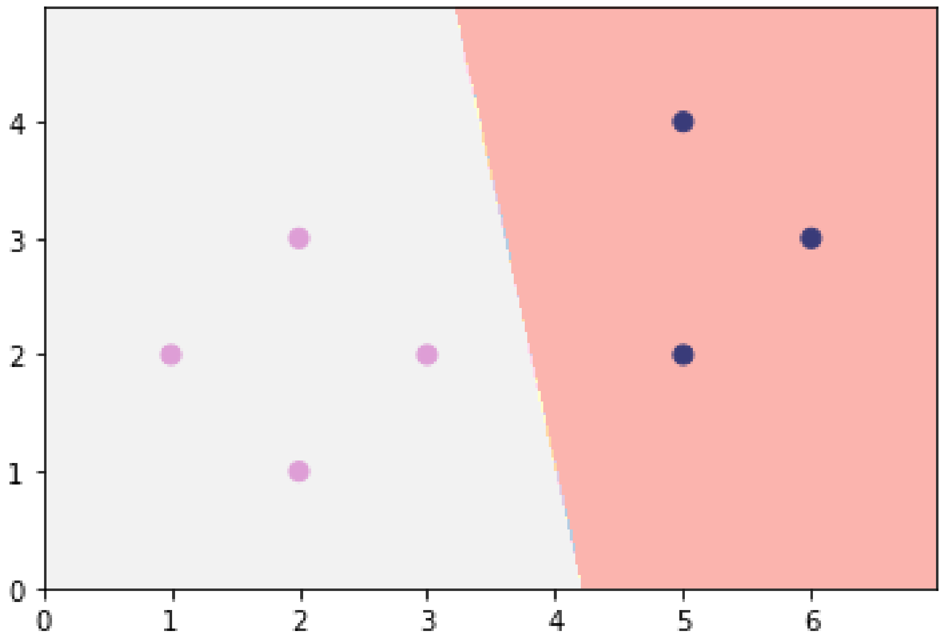
[2]:
```
from sklearn.linear_model import LinearRegression, RidgeClassifier
from sklearn.metrics import accuracy_score
```

(a) Prvo, isprobajte *ugraeni* model na linearno odvojivom skupu podataka `seven` ($N = 7$).

[3]:
```
seven_X = np.array([[2,1], [2,3], [1,2], [3,2], [5,2], [5,4], [6,3]])
seven_y = np.array([1, 1, 1, 1, 0, 0, 0])
```

[4]:
```
model = RidgeClassifier(alpha = 0).fit(seven_X, seven_y)
mlutils.plot_2d_clf_problem(seven_X, seven_y, h = model.predict)

y_pred = model.predict(seven_X)
acc = accuracy_score(y_pred, seven_y)
print(f"accuracy = {acc}")
```

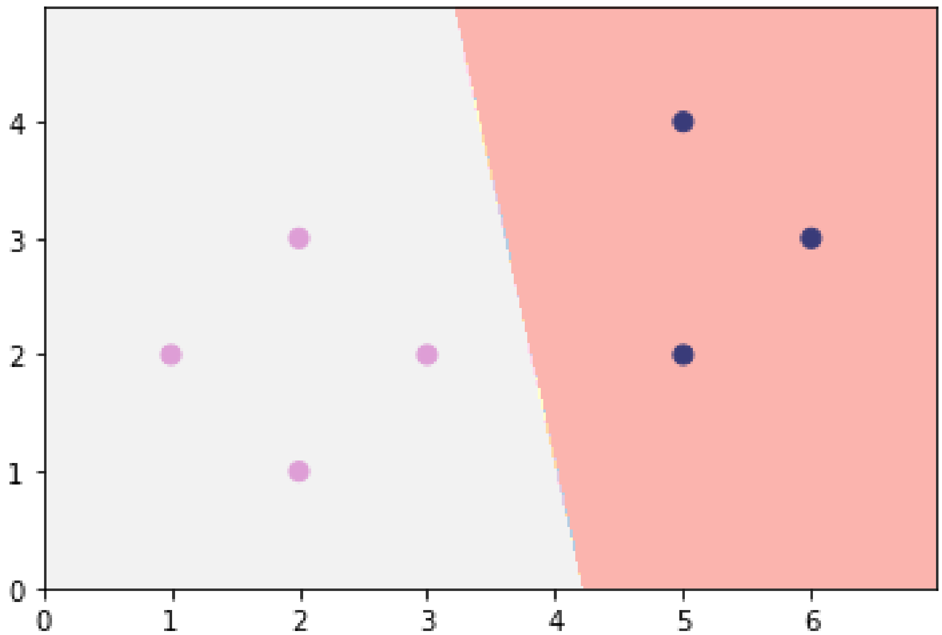
```
accuracy = 1.0
```



Kako bi se uvjerili da se u isprobanoj implementaciji ne radi o niemu doli o obinoj linearnoj regresiji, napiite kôd koji dolazi do jednakog rjeenja koritenjem iskljuivo razreda `LinearRegression`. Funkciju za predikciju, koju predajete kao trei argument `h` funkciji `plot_2d_clf_problem`, moete definirati lambda-izrazom: `lambda x : model.predict(x) >= 0.5`.

[5]:
```
model = LinearRegression().fit(seven_X, seven_y)
mlutils.plot_2d_clf_problem(seven_X, seven_y, h = lambda x : model.predict(x) >= 0.5)

y_pred = model.predict(seven_X)
y_pred = list(map(lambda x : 1 if x > 0.5 else 0, y_pred))
acc = accuracy_score(y_pred, seven_y)
print(f"accuracy = {acc}")
```

```
accuracy = 1.0
```



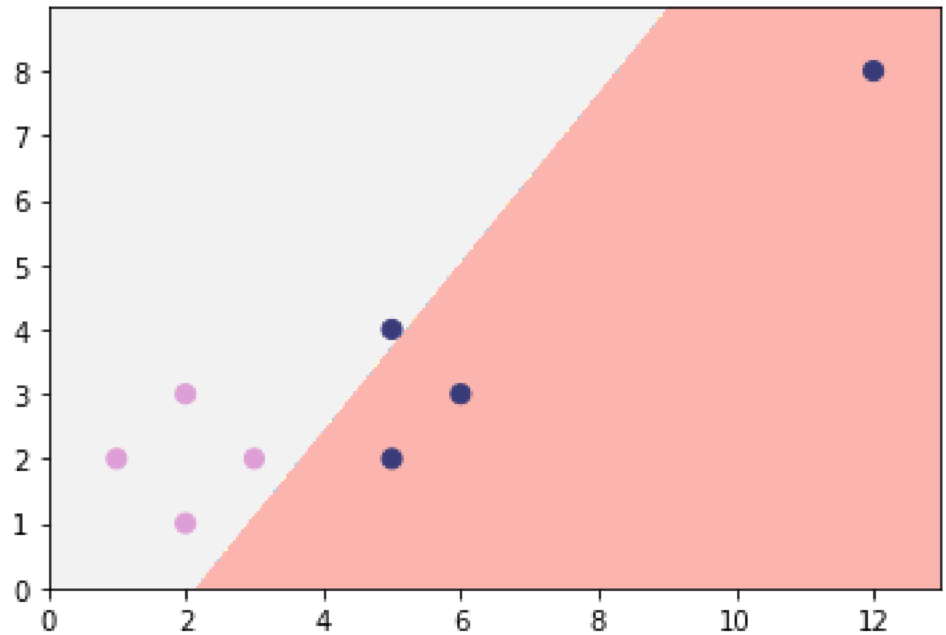
Q: Kako bi bila definirana granica izmeu klasa ako bismo koristili oznake klasa -1 i 1 umjesto 0 i 1?

(b) Probajte isto na linearno odvojivom skupu podataka `outlier` ($N = 8$):

[6]:
```
outlier_X = np.append(seven_X, [[12,8]], axis=0)
outlier_y = np.append(seven_y, 0)
```

[7]:
```
model = RidgeClassifier(alpha = 0).fit(outlier_X, outlier_y)
mlutils.plot_2d_clf_problem(outlier_X, outlier_y, h = model.predict)

y_pred = model.predict(outlier_X)
acc = accuracy_score(y_pred, outlier_y)
print(f"accuracy = {acc}")
```

```
accuracy = 0.875
```



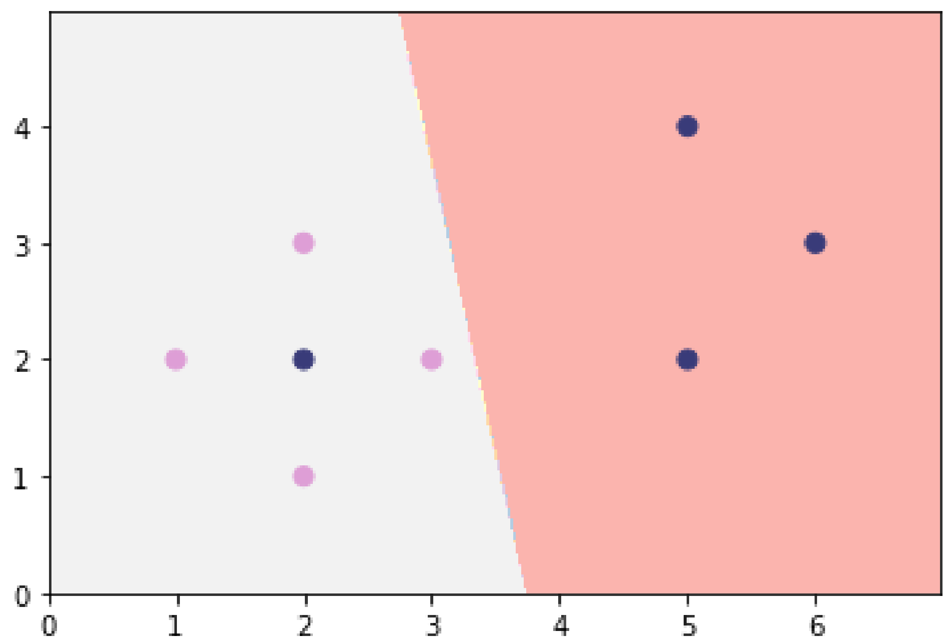
Q: Zato model ne ostvaruje potpunu tonost iako su podatci linearno odvojivi?

(c) Zavrno, probajte isto na linearno neodvojivom skupu podataka unsep ($N = 8$):

[8]:
```
unsep_X = np.append(seven_X, [[2,2]], axis=0)
unsep_y = np.append(seven_y, 0)
```

[9]:
```
model = RidgeClassifier(alpha = 0).fit(unsep_X, unsep_y)
mlutils.plot_2d_clf_problem(unsep_X, unsep_y, h = model.predict)

y_pred = model.predict(unsep_X)
acc = accuracy_score(y_pred, unsep_y)
print(f"accuracy = {acc}")
```

```
accuracy = 0.875
```



Q: Oito je zato model nije u mogunosti postii potpunu tonost na ovom skupu podataka. Meutim, smatrate li da je problem u modelu ili u podacima? Argumentirajte svoj stav.

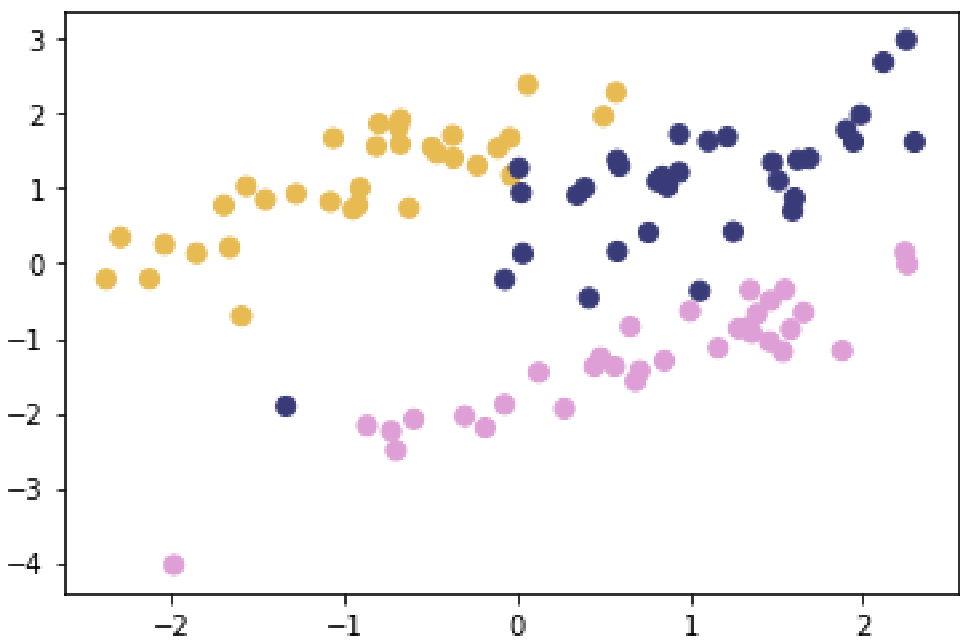
0.2.2 2. Vieklasna klasifikacija

Postoji vie naina kako se binarni klasifikatori mogu se upotrijebiti za vieklasnu klasifikaciju. Najee se koristi shema tzv. **jedan-naspram-ostali** (engl. *one-vs-rest*, OVR), u kojoj se trenira po jedan klasifikator h_j za svaku od K klasa. Svaki klasifikator h_j trenira se da razdvaja primjere klase j od primjera svih drugih klasa, a primjer se klasificira u klasu j za koju je $h_j(\mathbf{x})$ maksimalan.

Pomou funkcije `datasets.make_classification` generirajte sluajan dvodimenzijski skup podataka od tri klase i prikaite ga koristei funkciju `plot_2d_clf_problem`. Radi jednostavnosti, pretpostavite da nema redundantnih znaajki te da je svaka od klasa "zbijena" upravo u jednu grupu.

[10]:
```
from sklearn.datasets import make_classification
```

[11]:
```
X,y = make_classification(n_features = 2,
                    n_classes = 3,
                    n_redundant = 0,
                    n_clusters_per_class = 1)
mlutils.plot_2d_clf_problem(X, y)
```



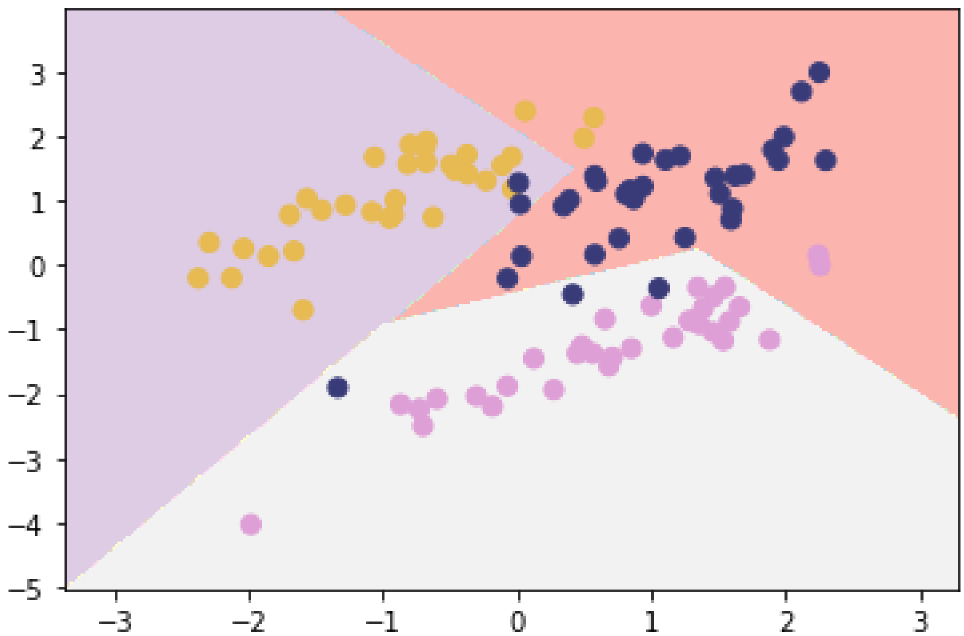
Trenirajte tri binarna klasifikatora, h_1, h_2 i h_3 te prikaite granice izmeu klasa (tri grafikona). Zatim definirajte $h(\mathbf{x}) = \mathrm{argmax}_j h_j(\mathbf{x})$ (napiite svoju funkciju `predict` koja to radi) i prikaite granice izmeu klasa za taj model. Zatim se uvjerite da biste identian rezultat dobili izravno primjenom modela `RidgeClassifier`, budui da taj model za vieklasan problem zapravo interno implementira shemu jedan-naspram-ostali.

Q: Alternativna shema jest ona zvana **jedan-naspram-jedan** (engl, *one-vs-one*, OVO). Koja je prednost sheme OVR nad shemom OVO? A obratno?

[12]:
```
models = []
for k in range(3):
    y_k = list(map(lambda y_i : 1 if y_i == k else 0, y))
    h = LinearRegression().fit(X, y_k)
    models.append(h)

def predict(X):
    pred = []
    for model in models:
        h = model.predict(X)
        h = map(lambda x : int(x > 0.5), h)
        pred.append(list(h))

    pred = np.array(pred)
    rows, cols = pred.shape
    classes = []
    for i in range(cols):
        xs = pred[:, i]
        k = max(enumerate(xs), key = lambda x : x[1])[0]
```

```
        classes.append(k)
    return np.array(classes)


mlutils.plot_2d_clf_problem(X, y, h = predict)

y_pred = model.predict(X)
acc = accuracy_score(y_pred, y)
print(f"accuracy = {acc}")
```

```
accuracy = 0.33
```



0.2.3 3. Logistika regresija

Ovaj zadatak bavi se probabilistikim diskriminativnim modelom, **logistikom regresijom**, koja je, unato nazivu, klasifikacijski model.

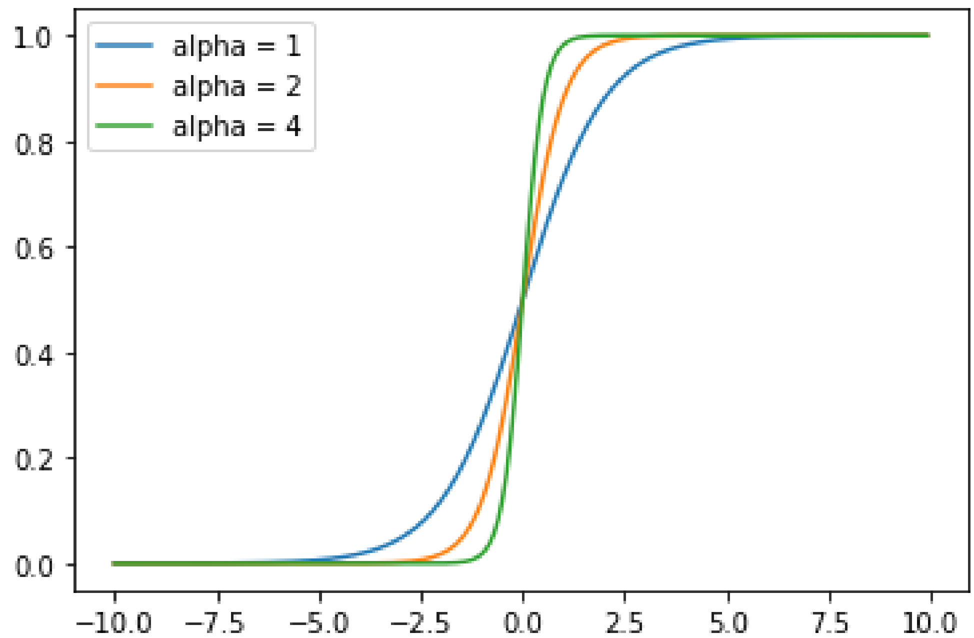
Logistika regresija tipian je predstavnik tzv. **poopenih linearnih modela** koji su oblika: $h(\mathbf{x}) = f(\mathbf{w}^\intercal\tilde{\mathbf{x}})$. Logistika funkcija za funkciju f koristi tzv. **logistiku** (sigmoidalnu) funkciju $\sigma(x) = \frac{1}{1+exp(-x)}$.

(a) Definirajte logistiku (sigmoidalnu) funkciju $\mathrm{sigm}(x) = \frac{1}{1+\exp(-\alpha x)}$ i prikaite je za $\alpha \in \{1, 2, 4\}$.

[13]:
```
def sigma(x, alpha = 1):
    return 1/(1 + np.exp(-alpha*x))
```

```python
xs = np.arange(-10,10,0.1)
plt.plot(xs, sigma(xs, 1), label = "alpha = 1")
plt.plot(xs, sigma(xs, 2), label = "alpha = 2")
plt.plot(xs, sigma(xs, 4), label = "alpha = 4")
plt.legend(loc = "best")
```

[13]:
```
<matplotlib.legend.Legend at 0x7f0c29fd1518>
```



Q: Zato je sigmoidalna funkcija prikladan izbor za aktivacijsku funkciju poopenoga linearnog modela?

Q: Kakav utjecaj ima faktor α na oblik sigmoide? to to znai za model logistike regresije (tj. kako izlaz modela ovisi o normi vektora teina $\mathbf{w}$)?

(b) Implementirajte funkciju

```
lr_train(X, y, eta=0.01, max_iter=2000, alpha=0, epsilon=0.0001,
trace=False)
```

za treniranje modela logistike regresije gradijentnim spustom (*batch* izvedba). Funkcija uzima oznaeni skup primjera za uenje (matrica primjera `X` i vektor oznaka `y`) te vraa $(n+1)$-dimenzijski vektor teina tipa `ndarray`. Ako je `trace=True`, funkcija dodatno vraa listu (ili matricu) vektora teina $\mathbf{w}^0, \mathbf{w}^1, \ldots, \mathbf{w}^k$ generiranih kroz sve iteracije optimizacije, od 0 do k. Optimizaciju treba provoditi dok se ne dosegne `max_iter` iteracija, ili kada razlika u pogreci unakrsne entropije izmeu dviju iteracija padne ispod vrijednosti `epsilon`. Parametar `alpha` predstavlja faktor L2-regularizacije.

Preporuamo definiranje pomone funkcije `lr_h(x,w)` koja daje predikciju za primjer `x` uz zadane teine `w`. Takoer, preporuamo i funkciju `cross_entropy_error(X,y,w)` koja izraunava pogreku unakrsne entropije modela na oznaenom skupu `(X,y)` uz te iste teine.

NB: Obratite pozornost na to da je nain kako su definirane oznake ($\{+1, -1\}$ ili $\{1, 0\}$) kompatibilan s izraunom funkcije gubitka u optimizacijskome algoritmu.

[47]:
```
from numpy import linalg
import pdb

def lr_h(X, w):
    logit  = (w @ X.T)
    return 1/(1+np.exp(-logit))

def cross_entropy_error(X, y, w, alpha):
    epsilon = 1e-5
    y_pred = lr_h(Fi, w).flatten()
    losses = -y*np.log(y_pred + epsilon) - (1-y)*np.log(1-y_pred + epsilon)
    norm = np.dot(w[0][1:], w[0][1:].T).item()
    return sum(losses)/len(y_pred) + alpha*norm/2

def lr_train(X, y, eta=0.01, max_iter=2000, trace=False, print_trace=False␣
 ↪,alpha=0, epsilon=0.000001):
    y_values = np.unique(y)
    if 1 in y_values and 0 in y_values:
        # Calculating grad for logistic reg. loss where y = {0,1}
        grad_calc = lambda X, y, w: (lr_h(X, w) - y) @ X
    elif 1 in y_values and -1 in y_values:
        # Calculating grad for logistic reg. loss where y = {-1,1}
        grad_calc = lambda X, y, w: -(y @ X)/(1 + exp(y @ (X @ w.T))).
 ↪reshape(1,-1)
    else:
        raise RuntimeError("Y values doesn't have known encoding")

    # 1 x 3
    w = np.zeros(len(X[-1])).reshape(1,-1)
    old_error = cross_entropy_error(X, y, w, alpha)
    ws = [w]
    for i in range(max_iter):
        w = ws[-1].copy()
        error = cross_entropy_error(X, y, w, alpha)
        if i % 100 == 0 and print_trace:
            print(f"iter={i}, error={error}")
        if abs(old_error - error) <= epsilon and i != 0:
            break
        else:
            old_error = error
        # y_pred : 1 x 7
        #y_pred = lr_h(X, w)

        # grad : 1 x 3
        grad = grad_calc(X = X, y = y, w = w).flatten()
```

```python
        # w : 1 x 3
        w[0,0]  = w[0,0] - eta*grad[0]
        w[0,1:] = w[0,1:]*(1-eta*alpha) - eta * grad[1:]
        if trace:
            ws.append(w)
        else:
            ws[-1] = w
    return ws
```

(c) Koristei funkciju `lr_train`, trenirajte model logistike regresije na skupu `seven`, prikaite dobivenu granicu izmeu klasa te izraunajte pogreku unakrsne entropije.

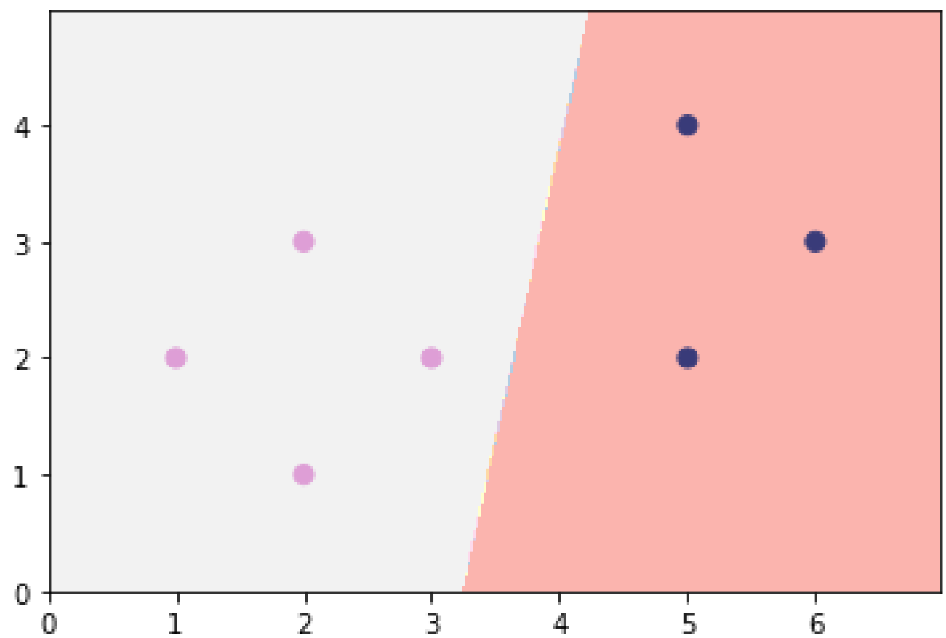
NB: Pripazite da modelu date dovoljan broj iteracija.

[82]:

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import log_loss

X = np.array([[2,1], [2,3], [1,2], [3,2], [5,2], [5,4], [6,3]])
y = np.array([1, 1, 1, 1, 0, 0, 0])
y_ = np.array([1, 1, 1, 1, -1, -1, -1])

Fi = PolynomialFeatures(1).fit_transform(X)
ws = lr_train(Fi, y, eta = 0.01, max_iter=3000, trace=True, print_trace=True, alpha = 0)
def predict_class(X):
    # Fi : 7 x 3
    Fi = PolynomialFeatures(1).fit_transform(X)
    # logits : 1 x 7
    logits = (Fi @ ws[-1].T).T
    # probs : 1 x 7
    probs = sigma(logits)
    classes = np.heaviside(probs - 0.5, 1)
    return classes

mlutils.plot_2d_clf_problem(X, y, h = predict_class)
```

```
iter=0, error=0.6931271807599427
iter=100, error=0.3943452615118996
iter=200, error=0.313654360047351
iter=300, error=0.26513594594461315
iter=400, error=0.23031662818784815
iter=500, error=0.20373467486659583
iter=600, error=0.18274929995545297
iter=700, error=0.16578161124349
iter=800, error=0.15179604860992932
iter=900, error=0.14007975299738182
iter=1000, error=0.13012621776936026
iter=1100, error=0.12156664138883237
```

```
iter=1200, error=0.11412688059278112
iter=1300, error=0.10759941225387025
iter=1400, error=0.10182455926340951
iter=1500, error=0.09667763295823882
iter=1600, error=0.09205995296087388
iter=1700, error=0.08789246329536142
iter=1800, error=0.08411112019374649
iter=1900, error=0.08066350987475893
iter=2000, error=0.07750633385289973
iter=2100, error=0.07460351520166275
iter=2200, error=0.07192475540559913
iter=2300, error=0.06944442238116004
iter=2400, error=0.0671406848187323
iter=2500, error=0.06499483179342015
iter=2600, error=0.06299073318835964
iter=2700, error=0.061114408195249
iter=2800, error=0.059353677532671005
iter=2900, error=0.057697881075248265
```



Q: Koji kriterij zaustavljanja je aktiviran?

Q: Zato dobivena pogreka unakrsne entropije nije jednaka nuli?

Q: Kako biste utvrdili da je optimizacijski postupak doista pronaao hipotezu koja minimizira pogreku uenja? O emu to ovisi?

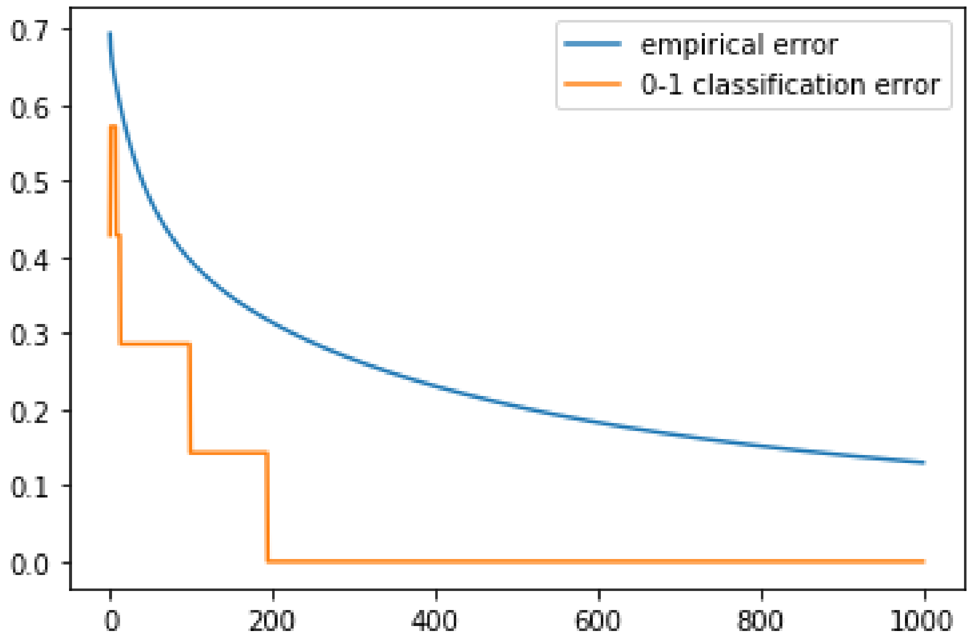
Q: Na koji nain biste preinaili kôd ako biste htjeli da se optimizacija izvodi stohastikim gradijentnim spustom (*online learning*)?

(d) Prikaite na jednom grafikonu pogreku unakrsne entropije (oekivanje logistikog gubitka) i pogreku klasifikacije (oekivanje gubitka 0-1) na skupu `seven` kroz iteracije optimizacijskog postupka. Koristite trag teina funkcije `lr_train` iz zadatka (b) (opcija `trace=True`). Na drugom grafikonu prikaite pogreku unakrsne entropije kao funkciju broja iteracija za razliite stope uenja, $\eta \in \{0.005, 0.01, 0.05, 0.1\}$.

[56]:
```
def predict(Fi, w):
    # Fi : 7 x 3
    # w : 1 x 3

    # logits : 1 x 7
    logits = (Fi @ w.T)
    probs = sigma(logits)
    classes = np.heaviside(probs - 0.5, 1)
    return classes.T
```

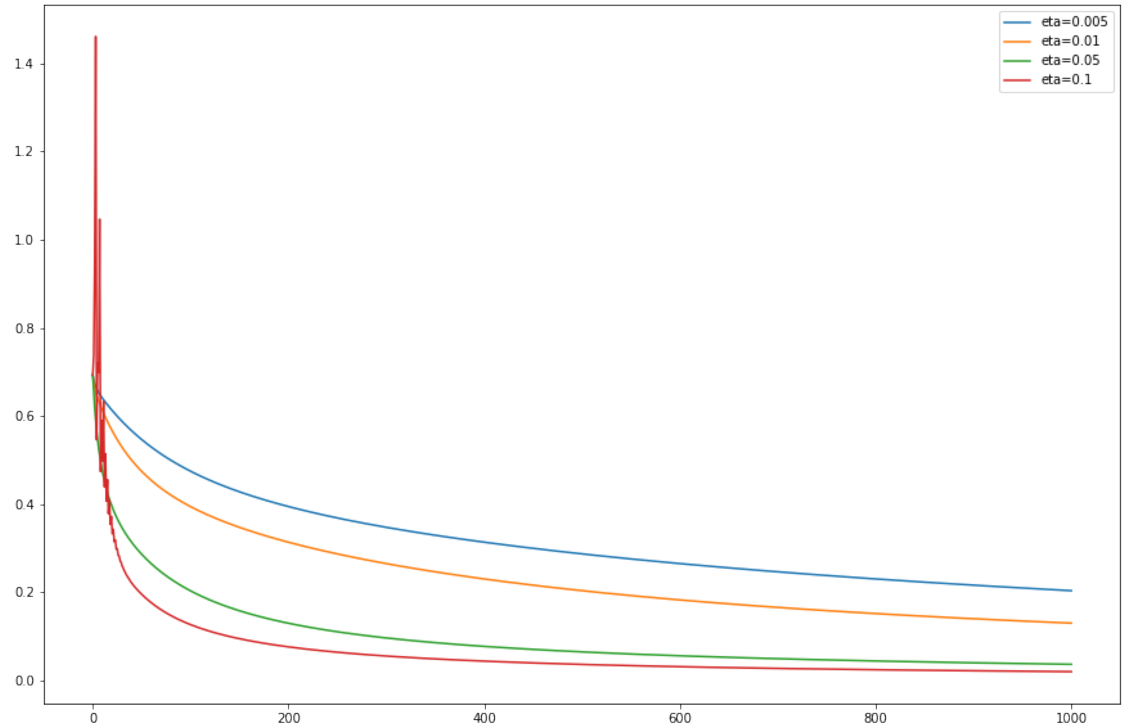
[57]:
```
X = np.array([[2,1], [2,3], [1,2], [3,2], [5,2], [5,4], [6,3]])
y = np.array([1, 1, 1, 1, 0, 0, 0])
Fi = PolynomialFeatures(1).fit_transform(X)

ws = lr_train(Fi, y, eta = 0.01, max_iter=1000, trace=True)
empirical_errors = []
class_errors = []
for w in ws:
    err = cross_entropy_error(Fi, y, w, alpha=0)
    empirical_errors.append(err)
    y_pred = predict(Fi, w)
    incorrect = abs((y_pred == y).astype(int) - 1)[0]
    class_errors.append(sum(incorrect) / 7)

plt.plot(empirical_errors, label = "empirical error")
plt.plot(class_errors, label = "0-1 classification error")
plt.legend(loc = "best")
```

[57]:
```
<matplotlib.legend.Legend at 0x7f0c2aa548d0>
```



[58]:
```python
etas = [0.005,0.01,0.05,0.1]
plt.figure(figsize=(15,10))

for eta in etas:
    ws = lr_train(Fi, y, eta = eta, max_iter=1000, trace=True, alpha = 0)
    errors = []
    for i in range(len(ws)):
        err = cross_entropy_error(Fi, y, ws[i],alpha = 0)
        errors.append(err)
    plt.plot(range(len(ws)), errors, label=f"eta={eta}")

plt.legend()
```

[58]: <matplotlib.legend.Legend at 0x7f0c29f32ac8>



Q: Zato je pogreka unakrsne entropije vea od pogreke klasifikacije? Je li to uvijek sluaj kod logistike regresije i zato?

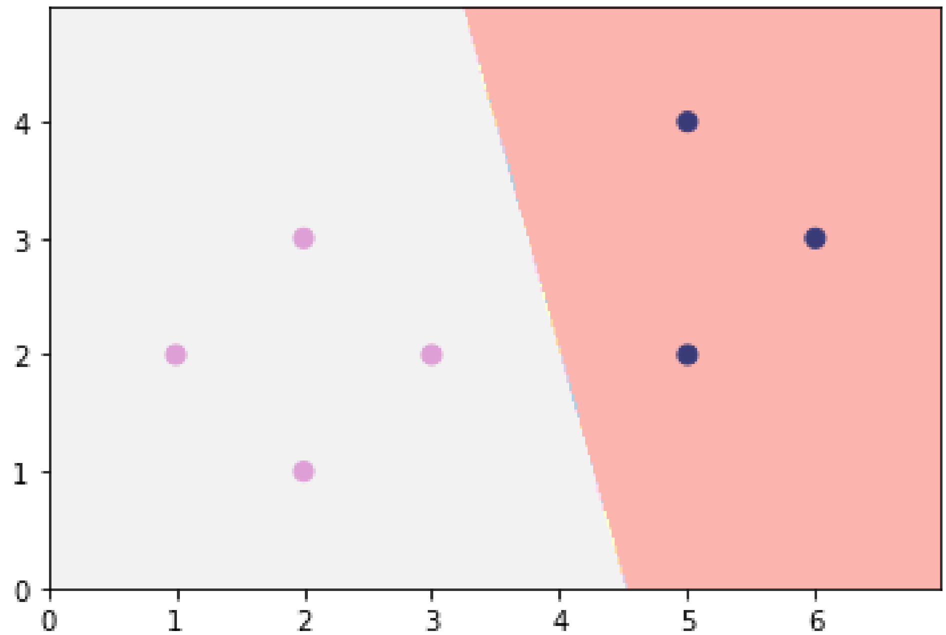
Q: Koju stopu uenja η biste odabrali i zato?

(e) Upoznajte se s klasom `linear_model.LogisticRegression` koja implementira logistiku regresiju. Usporedite rezultat modela na skupu `seven` s rezultatom koji dobivate pomou vlastite implementacije algoritma.

NB: Kako ugraena implementacija koristi naprednije verzije optimizacije funkcije, vrlo je vjerojatno da Vam se rjeenja nee poklapati, ali generalne performanse modela bi trebale. Ponovno, pripazite na broj iteracija i snagu regularizacije.

[59]:
```
from sklearn.linear_model import LogisticRegression
```

[60]:
```
model = LogisticRegression(solver = "lbfgs").fit(X, y)
mlutils.plot_2d_clf_problem(X, y, h = model.predict)
```

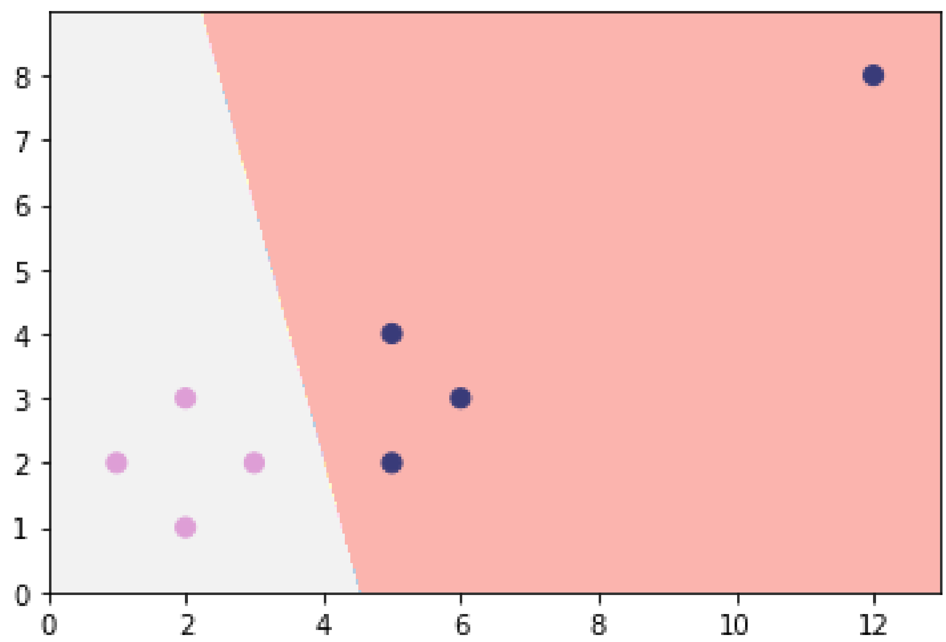


0.2.4 4. Analiza logistike regresije

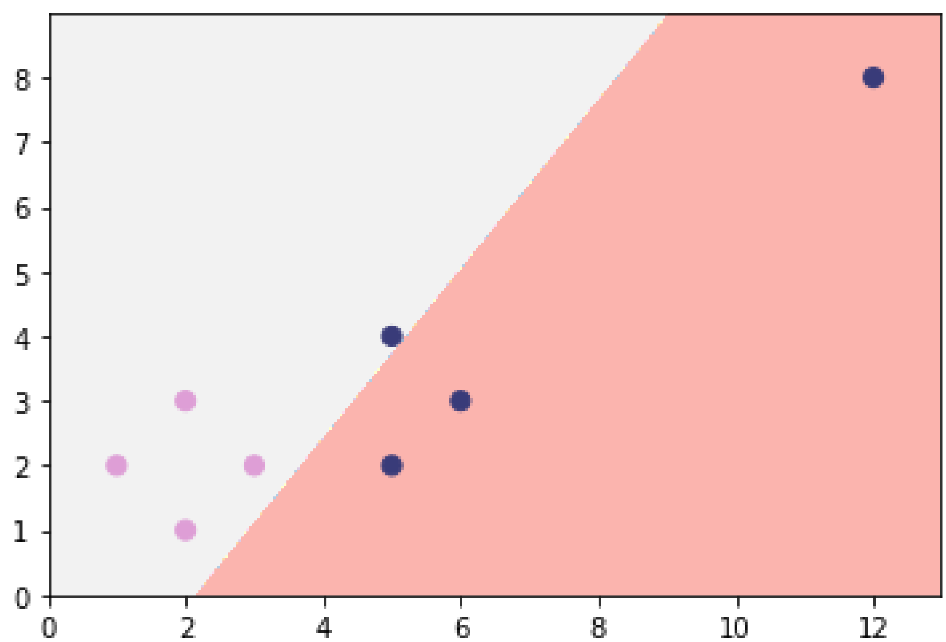
(a) Koristei ugraenu implementaciju logistike regresije, provjerite kako se logistika regresija nosi s vrijednostima koje odskau. Iskoristite skup `outlier` iz prvog zadatka. Prikaite granicu izmeu klasa.

Q: Zato se rezultat razlikuje od onog koji je dobio model klasifikacije linearnom regresijom iz prvog zadatka?

[61]:
```
#Logistic Regression
outlier_X = np.append(seven_X, [[12,8]], axis=0)
outlier_y = np.append(seven_y, 0)

model = LogisticRegression(solver = "lbfgs").fit(outlier_X, outlier_y)
mlutils.plot_2d_clf_problem(outlier_X, outlier_y, h = model.predict)
```



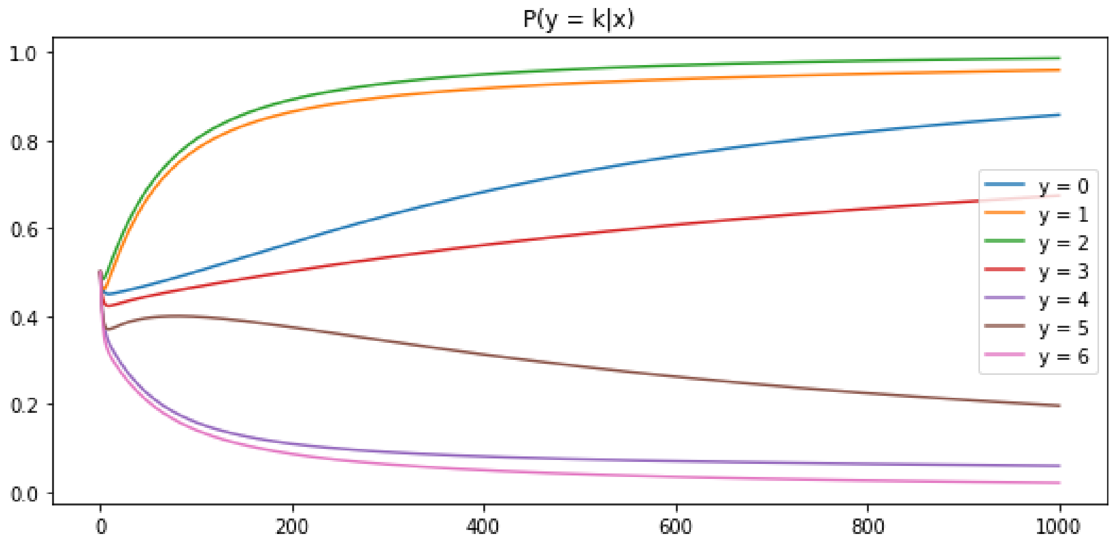
[62]:
```
#Linear regression
model = RidgeClassifier(alpha = 0).fit(outlier_X, outlier_y)
mlutils.plot_2d_clf_problem(outlier_X, outlier_y, h = model.predict)

y_pred = model.predict(outlier_X)
```

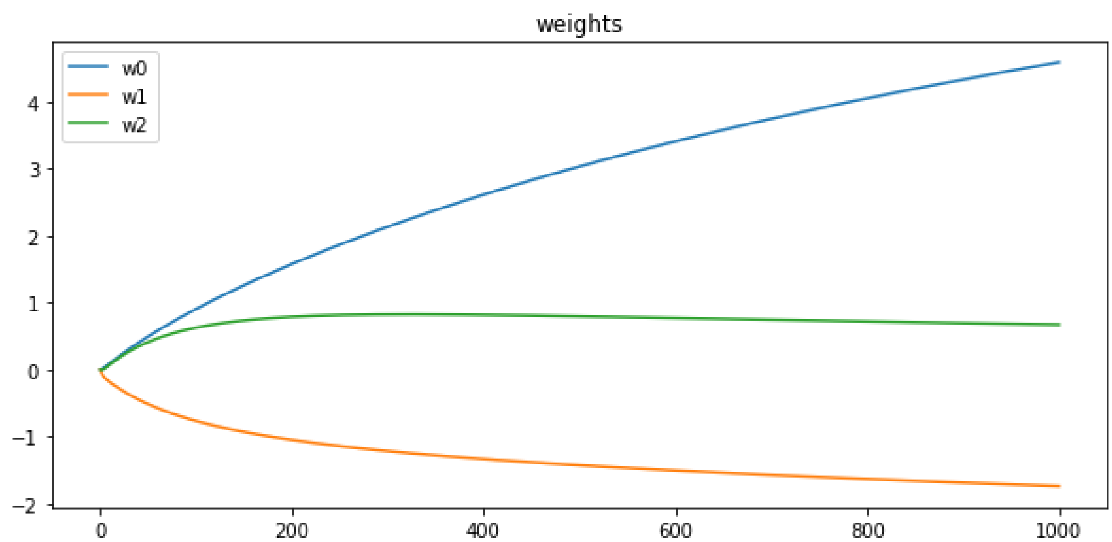


(b) Trenirajte model logistike regresije na skupu `seven` te na dva odvojena grafikona prikaite, kroz iteracije optimizacijskoga algoritma, (1) izlaz modela $h(\mathbf{x})$ za svih sedam primjera te (2) vrijednosti teina w_0, w_1, w_2.

[63]:
```
def draw_subplots(probs, weights):
    fig,ax = plt.subplots(2,1, figsize = (10,10))
    for i in range(7):
        ax[0].plot(probs[:, i], label = "y = " + str(i))
        ax[0].title.set_text("P(y = k|x)")
    ax[0].legend(loc = "best")

    ax[1].plot(weights[:, 0], label = "w0")
    ax[1].plot(weights[:, 1], label = "w1")
    ax[1].plot(weights[:, 2], label = "w2")
    ax[1].title.set_text("weights")
    ax[1].legend(loc = "best")
```

[72]:
```
seven_X = np.array([[2,1], [2,3], [1,2], [3,2], [5,2], [5,4], [6,3]])
seven_y = np.array([1, 1, 1, 1, 0, 0, 0])
Fi = PolynomialFeatures(1).fit_transform(seven_X)

ws = lr_train(Fi, seven_y, eta = 0.01, max_iter=1000, trace=True, alpha = 0)
probs = []
weights = []
for w in ws:
    y_pred = lr_h(Fi, w)
    probs.append(y_pred[0])
    weights.append(w[0])

probs   = np.array(probs)
weights = np.array(weights)

draw_subplots(probs, weights)
```



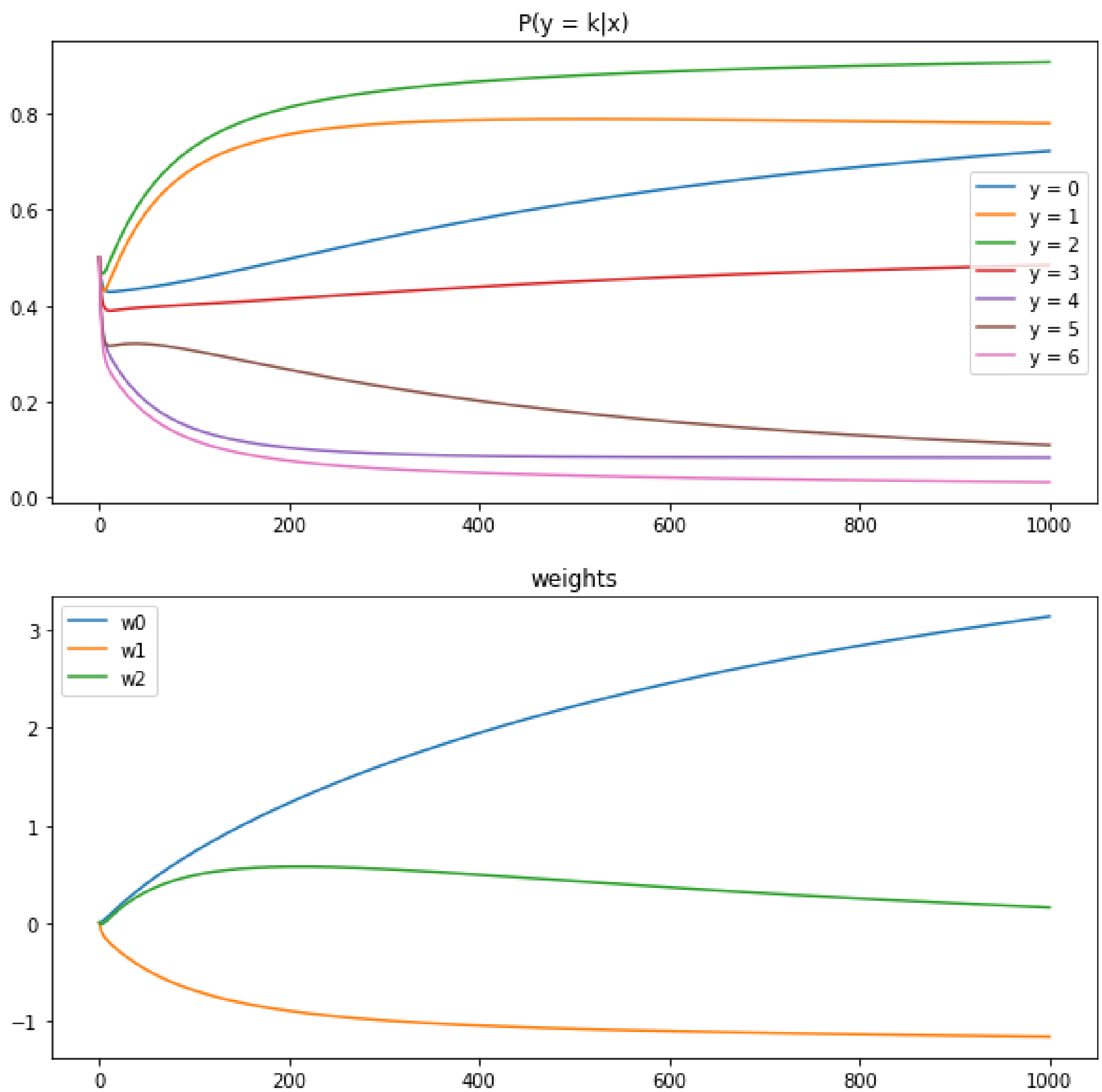


(c) Ponovite eksperiment iz podzadatka (b) koristei linearno neodvojiv skup podataka `unsep` iz prvog zadatka.

Q: Usporedite grafikone za sluaj linearno odvojivih i linearno neodvojivih primjera te komentirajte razliku.

[73]:
```
unsep_X = np.append(seven_X, [[2,2]], axis=0)
unsep_y = np.append(seven_y, 0)
Fi = PolynomialFeatures(1).fit_transform(unsep_X)

ws = lr_train(Fi, unsep_y, eta = 0.01, max_iter=1000, trace=True, alpha = 0)
weights = []
probs   = []
for w in ws:
```

```
    y_pred = lr_h(Fi, w)
    probs.append(y_pred[0])
    weights.append(w[0])

probs   = np.array(probs)
weights = np.array(weights)

draw_subplots(probs, weights)
```



0.2.5 5. Regularizirana logistika regresija

Trenirajte model logistike regresije na skupu seven s razliitim faktorima L2-regularizacije, $\alpha \in \{0, 1, 10, 100\}$. Prikaite na dva odvojena grafikona (1) pogreku unakrsne entropije te (2) L2-normu vektora **w** kroz iteracije optimizacijskog algoritma.

Q: Jesu li izgledi krivulja oekivani i zato?

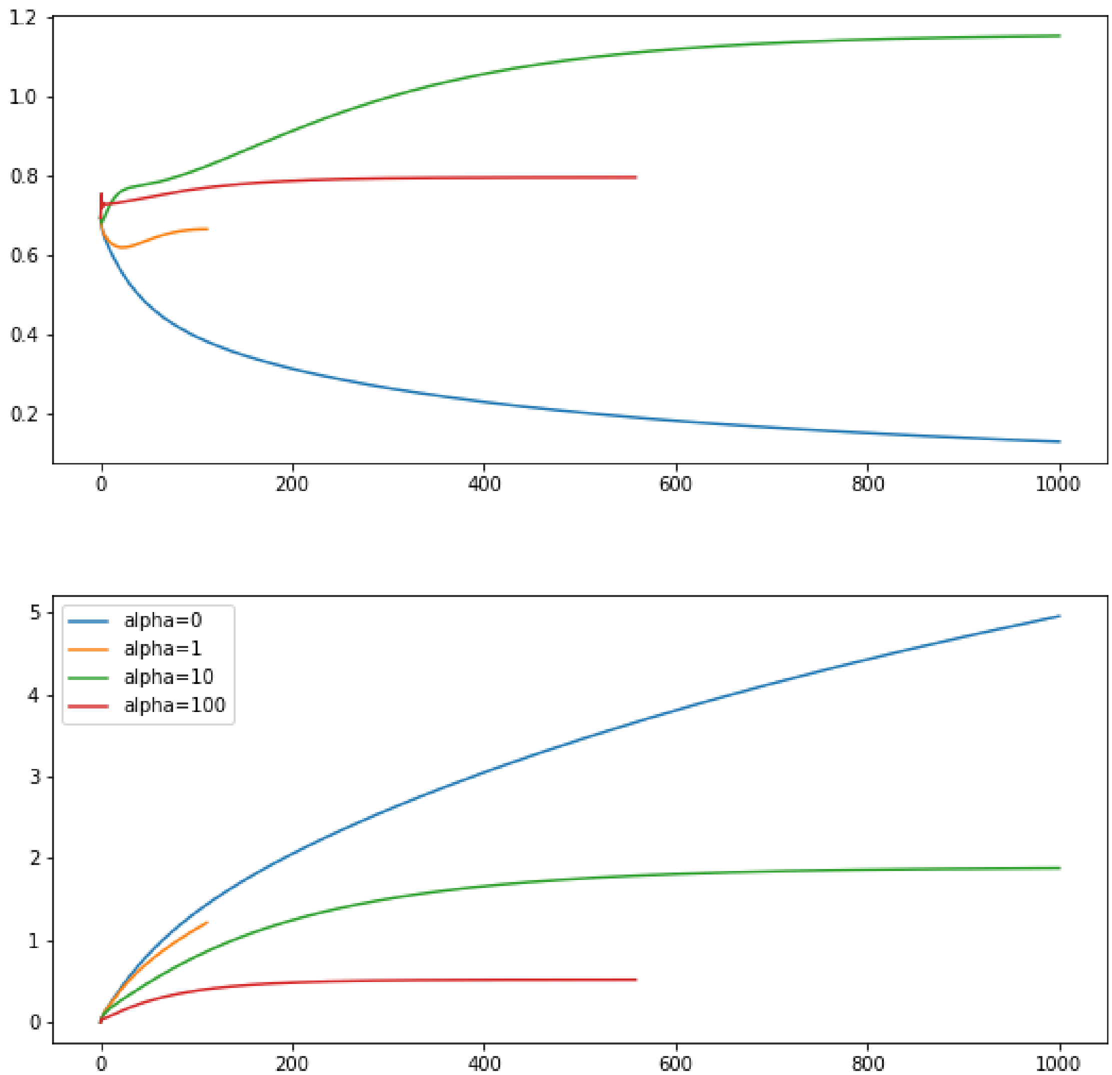
Q: Koju biste vrijednost za α odabrali i zato?

[27]:
```python
from numpy.linalg import norm
from math import sqrt
```

[75]:
```python
seven_X = np.array([[2,1], [2,3], [1,2], [3,2], [5,2], [5,4], [6,3]])
seven_y = np.array([1, 1, 1, 1, 0, 0, 0])
Fi = PolynomialFeatures(1).fit_transform(seven_X)

fig,axs = plt.subplots(2,1, figsize = (10,10))
fig.subplots_adjust(wspace = 0.3,hspace = 0.3)

alphas = [0, 1, 10, 100]

for i,alpha in enumerate(alphas):
    ws = lr_train(Fi, seven_y, eta = 0.01, max_iter=1000, trace=True, alpha =␣
 ↪alpha)
    cross_entropy = []
    norm = []
    for w in ws:
        cross_entropy.append(cross_entropy_error(Fi, seven_y, w, alpha))
        norm.append(sqrt(w.dot(w.T)))
    axs[0].plot(cross_entropy, label = f"alpha={alpha}")
    axs[1].plot(norm, label = f"alpha={alpha}")

_ = fig.suptitle('CROSS ENTROPY      -      L2 NORM', fontsize = 20)
plt.legend(loc = "best")
```

[75]:
```
<matplotlib.legend.Legend at 0x7f0c29c60c88>
```

CROSS ENTROPY - L2 NORM

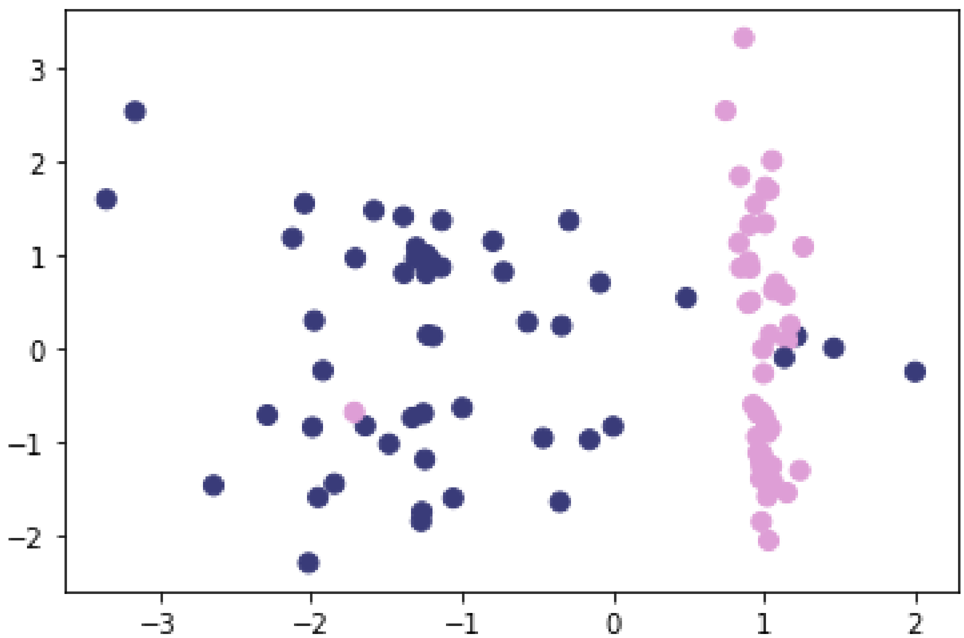


0.2.6 6. Logistika regresija s funkcijom preslikavanja

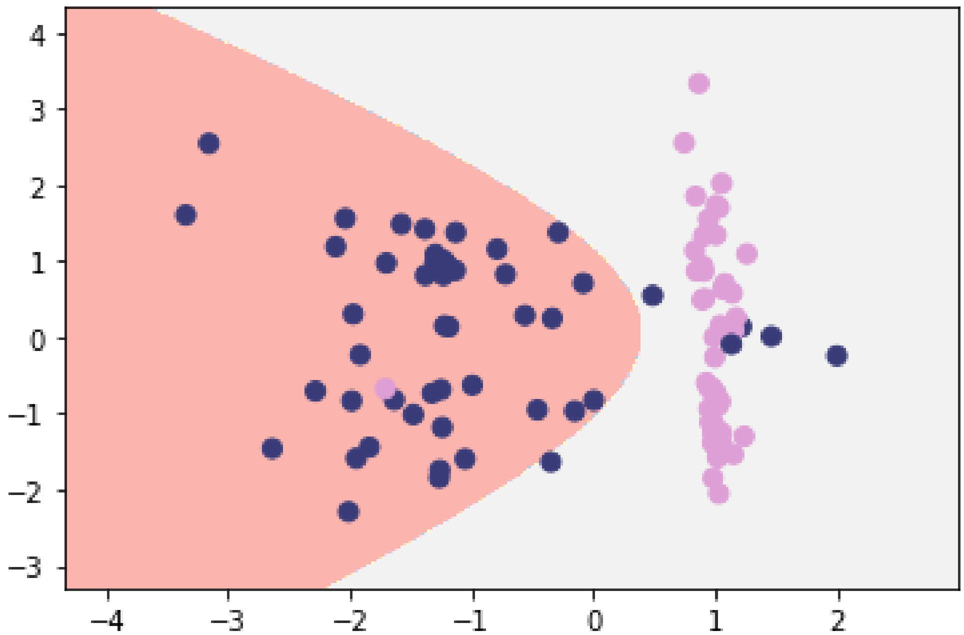
Prouite funkciju `datasets.make_classification`. Generirajte i prikaite dvoklasan skup podataka s ukupno $N = 100$ dvodimenzijskih ($n = 2$) primjera, i to sa dvije grupe po klasi (`n_clusters_per_class=2`). Malo je izgledno da e tako generiran skup biti linearno odvojiv, meutim to nije problem jer primjere moemo preslikati u viedimenzijski prostor znaajki pomou klase `preprocessing.PolynomialFeatures`, kao to smo to uinili kod linearne regresije u prvoj laboratorijskoj vjebi. Trenirajte model logistike regresije koristei za preslikavanje u prostor znaajki polinomijalnu funkciju stupnja $d = 2$ i stupnja $d = 3$. Prikaite dobivene granice izmeu klasa. Moete koristiti svoju implementaciju, ali se radi brzine preporua koristiti linear_model.LogisticRegression. Regularizacijski faktor odaberite po elji.

NB: Kao i ranije, za prikaz granice izmeu klasa koristite funkciju plot_2d_clf_problem. Funkciji kao argumente predajte izvorni skup podataka, a preslikavanje u prostor znaajki napravite unutar poziva funkcije h koja ini predikciju, na sljedei nain:

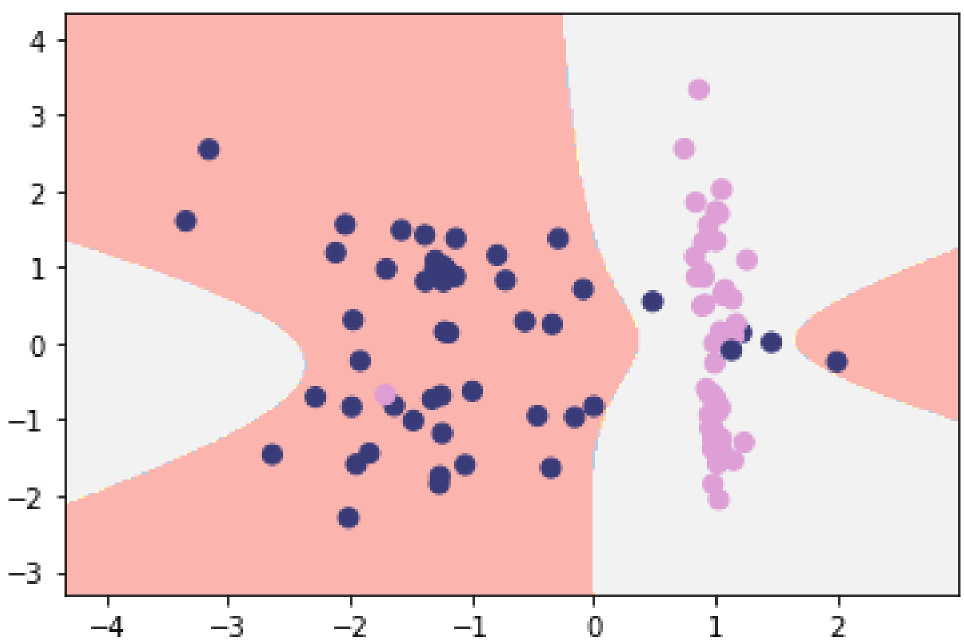
[29]:
```
from sklearn.preprocessing import PolynomialFeatures

X,y = make_classification(n_samples = 100,
                          n_features = 2,
                          n_classes = 2,
                          n_redundant = 0,
                          n_clusters_per_class = 2)

mlutils.plot_2d_clf_problem(X, y)
```



[30]:
```
Fi_2 = PolynomialFeatures(2).fit_transform(X)
Fi_3 = PolynomialFeatures(3).fit_transform(X)

model = LogisticRegression(solver = "lbfgs", penalty = "l2").fit(Fi_2, y)
mlutils.plot_2d_clf_problem(X, y, h = lambda x : model.
 ↪predict(PolynomialFeatures(2).fit_transform(x)))
```



[31]:
```
model = LogisticRegression(solver = "lbfgs", penalty = "l2").fit(Fi_3, y)
mlutils.plot_2d_clf_problem(X, y, h = lambda x : model.
 ↪predict(PolynomialFeatures(3).fit_transform(x)))
```



Q: Koji biste stupanj polinoma upotrijebili i zato? Je li taj odabir povezan s odabirom regularizacijskog faktora α? Zato?