Regresijske tehnike kao dio nadziranog učenja i Scikit-learn Python modul

Mijošević, Tomislav

Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku

Tomislav Mijošević

Regresijske tehnike kao dio nadziranog učenja i scikit-learn Python modul

Završni rad

Osijek, 2018.

Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku

Tomislav Mijošević

Regresijske tehnike kao dio nadziranog učenja i scikit-learn Python modul

Završni rad

Mentor: izv. prof. dr. sc. Domagoj Matijević

Osijek, 2018.

Sadržaj

Sažetak

U ovom radu upoznat ćemo se s glavnim problemima nadziranog učenja: regresijom i klasifikacijom. Glavni dio ovog rada podijeljen je na dva poglavlja, pri čemu se prvo bavi regresijskim problemima, a drugo klasifikacijskim. Pokazat ćemo i objasniti konstrukcije matematičkih modela potrebnih za riješavanje ovih problema, ukazati na nedostatke osnovnih modela i pokazati način na koji te nedostatke možemo ukloniti. Na kraju rada pokazat ćemo implementaciju tih modela u programskom jeziku Python pomoću modula scikit-learn.

Ključne riječi

nadzirano učenje, linearna regresija, polinomijalna regresija, logistička regresija, scikit-learn

Abstract

In this paper we will introduce main problems of supervised learning: regression and classification. Main part of the paper is divided into two sections, whereby first deals with regression problems, and the second deals with classification problems. We will show and explain the construction of mathematical models needed to solve these problems, point out the shortcomings of the basic models and show how to remove those shortcomings. At the end of the paper we will show the Python implementation of these models using scikit-learn module.

Key words

supervised learning, linear regression, polynomial regression, logistic regression, scikit-learn

1 Uvod

Strojno učenje dijelimo na nadzirano i nenadzirano učenje. Kod nadziranog učenja početni skup podataka se sastoji od parova (x, y) gdje je x ulazna varijabla ili ulazni vektor, a y je varijabla "odgovor" ili vektor "odgovora". Zadatak algoritma je odrediti zavisnost između varijabli x i y te na temelju toga odrediti funkciju $\hat{y} = f(x)$(funkciju f nazivamo estimator ili procjenitelj). Pomoću te funkcije predviđamo podatke za koje nemamo točna mjerenja. Problemi nadziranog učenja dijele se na regresijske i klasifikacijske probleme. U regresijskim problemima zavsnost varijabli pokušavamo opisati neprekidnom funkcijom, dok kod klasifikacijskih problema ulazne podatke pokušavamo razvrstati u diskretne kategorije.
U prvom poglavlju rada bavit ćemo se linearnom regresijom. Prvo ćemo pokazati model linearne regresije jedne varijable, a potom ćemo model proširiti na funkcije više varijabli, ali i na neke nelinearne funkcije. Na kraju poglavlja pokazat ćemo neke od metoda optimizacije algoritma linearne regresije.
U drugom poglavlju bavit ćemo se jednim od najčešće korištenih algoritama strojnog učenja: logističkom regresijom. Pokazat ćemo konstrukciju modela logističke regresije za binarne probleme, a na kraju poglavlja izvest ćemo poopčenje tog modela za proizvoljan broj klasa. Na kraju rada pokazat ćemo i objasniti implementaciju ovih modela na osnovnim podatcima iz scikit-learn Python modula. U nastavku rada koristit ćemo sljedeću notaciju:

x - ulazna varijabla ili vektor svojstava
y - izlazna/"ciljna" varijabla
m - broj primjera za učenje
(x, y) - jedan primjer za učenje
$(x^{(i)}, y^{(i)})$ - i-ti primjer za učenje
n - broj promatranih svojstava
$x^{(i)}$ - svojstva i-tog trening primjera
$x_j^{(i)}$ - vrijednost j-tog svojstva u i-tom primjeru
$h_\theta(x)$ - funkcija procjene

2 Linearna regresija

2.1 Linearna regresija jedne varijable

2.1.1 Funkcija cijene

Promotrimo sljedeći skup podataka:

Kvadratura kuće (kvadratne stope)	**Cijena kuće ($)**
2000	460000
1521	232000
1684	315000
938	178000
⋮	⋮

Tablica 2.1

Želimo odrediti matematički model pomoću kojega ćemo moći pretpostaviti cijenu kuće, ako znamo njenu kvadraturu. Pretpostavimo da cijena ovisi linearno o kvadraturi kuće tj. $h_\theta(x) = \theta_0 + \theta_1 x$, gdje su θ_0 i θ_1 parametri modela. Cilj je pronaći θ_0 i θ_1 takve da dobiveni pravac što bolje aproksimira podatke koje smo dobili mjerenjem. Formalno, želimo pronaći

$$\min_{\theta_0,\theta_1} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2. \tag{2.1}$$

Radi jednostavnosti definirat ćemo funkciju cijene sljedećim izrazom

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2. \tag{2.2}$$

Sada izraz (2.1) možemo lakše zapisati

$$\min_{\theta_0,\theta_1} J(\theta_0, \theta_1). \tag{2.3}$$

Funkcija $J(\theta_0, \theta_1)$ naziva se još i **funkcija kvadratne pogreške**.

2.1.2 Metoda najbržeg silaska

Metoda najbržeg silaska[1] jedan je od najčešće primjenjivanih algoritama optimizacije u strojnom učenju. Promotrimo prvo primjer minimizacije na proizvoljnoj funkciji $F(x_0, x_1)$. Prvo trebamo odabrati početne vrijednosti za x_1 i x_2 (najčešće se uzima $x_0 = x_1 = 0$). Nakon toga ponavljamo sljedeći korak dok ne dođemo do konvergencije

$$x_j = x_j - \alpha \frac{\partial}{\partial x_j} F(x_0, x_1) \quad (za\ j = 0, 1). \tag{2.4}$$

Prilikom implementacije treba paziti da se varijable ažuriraju simultano, inače može doći do nepredviđenog ponašanja algoritma.

[1] U nastavku ćemo koristiti oznaku MNS.

Dobra implementacija
$temp_0 := x_0 - \alpha \frac{\partial}{\partial x_0} F(x_0, x_1)$
$temp_1 := x_1 - \alpha \frac{\partial}{\partial x_1} F(x_0, x_1)$
$x_0 := temp_0$
$x_1 := temp_1$

Loša implementacija
$temp_0 := x_0 - \alpha \frac{\partial}{\partial x_0} F(x_0, x_1)$
$x_0 := temp_0$
$temp_1 := x_1 - \alpha \frac{\partial}{\partial x_1} F(x_0, x_1)$
$x_1 := temp_1$

Za MNS jako je bitno dobro izabrati stopu učenja α. Ako je α previše mali MNS će jako sporo konvergirati, a ako uzmemo previše velik α MNS će divergirati. Postoje algoritmi za ažuriranje parametra α pri svakoj iteraciji MNS, ali konvergencija prema lokalnom minimumu je moguća i za fiksan α.

Za primjenu MNS na model linearne regresije trebamo još samo izračunati parcijalne derivacije funkcije $J(\theta_0, \theta_1)$.

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \tag{2.5}$$
$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

Sada parcijalne derivacije uvrstimo u (2.4) i taj postupak ponavljamo dok ne dođemo do konvergencije (često se za uvjet konvergencije uzima $\Delta J < 0.00001$). Uvrštavanjem dobijenih koeficijenata θ_0 i θ_1 u funkciju $h_\theta(x)$ dobijemo pravac koji najbolje aproksimira početne podatke.

2.2 Linearna regresija više varijabli

2.2.1 Vektorizacija

Često se problem ne može precizno modelirati samo jednom varijablom. Promotrimo sljedeći slučaj

Kvadratura kuće (kvadratne stope)	**Broj soba**	**Broj katova**	**Starost kuće**	**Cijena kuće ($)**
2000	5	1	45	460000
1521	3	2	20	232000
1684	3	2	30	315000
938	2	1	36	178000
⋮	⋮	⋮	⋮	⋮

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

Radi jednostavnijeg zapisa definirat ćemo $x_0^{(i)} = 1 \quad \forall\, i = 1...m$.
Sada uz oznake

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \tag{2.6}$$

funkciju $h_\theta(x)$ možemo kraće zapisati na sljedeći način

$$h_\theta(x) = \theta^T x. \tag{2.7}$$

Popularni programski jezici danas imaju dobre pakete za numeričku linearnu algebru. Ti paketi često koriste paralelizam za računanje s vektorima pa će MNS za funkcije više varijabli raditi puno efikasnije ako implementiramo vektorski oblik

$$\theta = \theta - \alpha\delta, \text{ gdje je } \delta = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}. \tag{2.8}$$

Drugi način minimizacije je metoda **normalnih jednadžbi**

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}, y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$$\theta = (X^T X)^\dagger X^T y. \tag{2.9}$$

Za ovu metodu nije potrebno tražiti parametar α. Još jedna prednost ove metode je činjenica da rješenje određujemo analitički, a ne iterativno. Kako je za većinu implementacija cijena invertiranja matrice $X^T X$ klase $\mathbf{O}(n^3)$, ova će metoda biti jako spora za modele sa puno svojstava pa je za $n > 1000$ bolje koristiti MNS.

2.2.2 Transformacija svojstava

Drugi način za ubrzavanje MNS na velikim skupovima podataka je **transformacija podataka**[2]. Želimo transformirati podatke tako da sva svojstva poprimaju slične raspone vrijednosti. Pogledajmo podatke iz prethodnog primjera:

$$x_1 = \text{kvadratura(0-2000)}$$
$$x_2 = \text{broj soba(1-5)}$$

Ovakva razlika u rasponima često dovede do znatnog povećanja broja potrebnih iteracija MNS. Najčešći oblici transformacije svojstava su **min-max normalizacija**(2.10) i **standardizacija**(2.11)

$$\hat{x}_j^{(i)} = \frac{x_j^{(i)} - \min\limits_i(x_j^{(i)})}{\max\limits_i(x_j^{(i)}) - \min\limits_i(x_j^{(i)})} \tag{2.10}$$

$$\hat{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j} \tag{2.11}$$

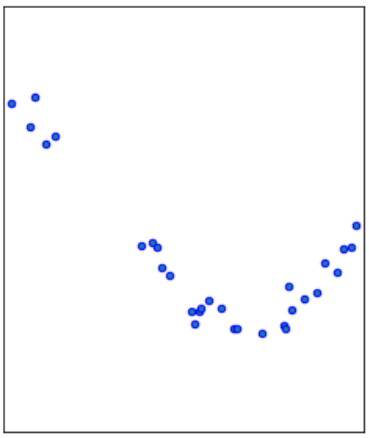
[†] pseudo-inverz matrice

[2] eng. *feature scaling*

gdje je μ_j matematičko očekivanje svojstva j, a σ_j standardna devijacija. Primjenimo li standardizaciju na prethodne podatke dobit ćemo $-1 \leq x_1 \leq 1$ i $-1 \leq x_2 \leq 1$. Transformacija podataka nije potrebna ako za minimizaciju koristimo metodu normalnih jednadžbi.

2.2.3 Polinomijalna regresija

Može se dogoditi da imamo model koji ovisi o jednoj varijabli, ali je aproksimacija pomoću modela linearne regresije jedne varijabe loša. U takvim slučajevima podatke možemo pokušati aproksimirati polinomima višeg stupnja, trigonometrijskim funkcijama, logaritamskom funkcijom...
Iako postoje nelinearni regresijski algoritmi, oni su memorijski i vremenski puno zahtjevniji pa je bolje ovakve probleme prvo pokušati modelirati polinomijalnom regresijom. Pogledajmo sljedeće podatke:



Slika 2.1

Očito je da bi polinom višeg stupnja puno bolje opisao ove podatke od pravca, ali kako konstruirati takav polinom?
Pretpostavimo da podatke želimo aproksimirati polinomom drugog stupnja. Tada će funkcija procjene biti sljedećeg oblika:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2. \tag{2.12}$$

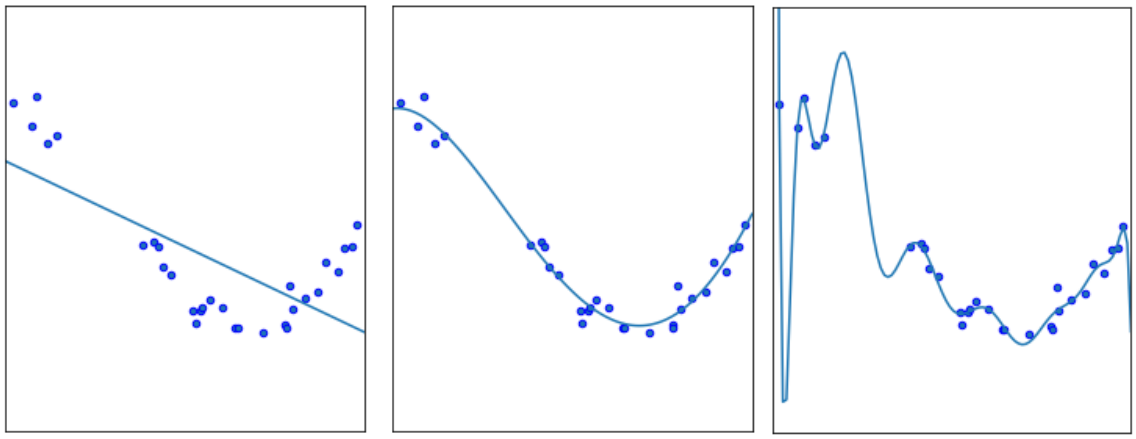
Bitno je uočiti da je ovo još uvijek linearan model. Uvedemo li oznaku $z = (1, x, x^2)^T$ funkcija procjene sada postaje

$$h_\theta(x) = \theta_0 z_0 + \theta_1 z_1 + \theta_2 z_2 = \theta^T z \tag{2.13}$$

što upravo odgovara obliku linearne regresije više varijabli (2.7).

2.2.4 Underfitting i overfitting

Za dobru aproksimaciju jako je bitan izbor promatranih svojstava. Uzmemo li premalo svojstava doći će do "*underfittinga*" tj. dobit ćemo previše jednostavnu funkciju koja nije dobar procjenitelj čak ni za zadane podatke. Primjetimo li da se dogodio underfitting, potrebno je povećati broj promatranih svojstava, ali pri tome treba biti oprezan, jer će previše svojstava predstavljati problem za određivanje dobre aproksimacije. Uzmemo li previše svojstava može se pojaviti problem "*overfittinga*". Funkcija procjene će jako dobro aproksimirati početne podatke ($J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), ali će loše aproksimirati nove primjere. Pogledajmo primjer underfitting i overfittinga na podatcima sa slike 2.1



(a) Underfit
polinom 1. stupnja

(b) Dobra aproksimacija
polinom 4. stupnja

(c) Overfit
polinom 15. stupnja

Slika 2.2: Aproksimacije podataka polinomima različitih stupnjeva

Ako promatramo jednodimenzionalne ili dvodimenzionalne podatke funkciju procjene možemo grafički prikazati pa je uočavanje i rješavanje overfittinga puno jednostavnije nego kod problema viših dimenzija. Kod takvih problema overfitting rješavamo na jedan od sljedećih načina:

1. Redukcija broja svojstava
   - Sami odaberemo koja svojstva odbaciti
   - Algoritam za izbor svojstava
2. Regularizacija
   - Zadržimo sva svojstva, ali ograničimo vrijednosti parametara θ_j

Redukciju koristimo ako nismo sigurni jesu li sva svojstva bitna za model ili ako imamo svojstva koja su međusobno zavisna, dok ćemo regularizaciju koristiti ako su sva svojstva bitna za procjenu. Postupak regularizacije provodimo izmjenom funkcije cjene J na sljedeći način:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]. \tag{2.14}$$

Za ovako definiranu funkciju J moramo izmjeniti i implementaciju MNS

$$\theta_0 = \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$
$$\theta_j = \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \tag{2.15}$$

Treba uočiti da se parametar θ_0 ne regularizira.
Regularizacije se može primjeniti i na metodu normalnih jednadžbi na sljedeći način

$$A = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$$

$$\theta = (X^TX + \lambda A)^{-1}X^Ty \tag{2.16}$$

Može se pokazati da će matrica $(X^TX + \lambda A)$ uvijek biti invertibilna za $\lambda > 0$ pa ovdje ne moramo koristiti pseudo-inverz.
Ovako definiran algoritam naziva se l_2 regularizacija s parametrom regularizacije λ ili linearna regresija s l_2 penalizacijom (*eng. ridge regression*).
Često se koristi i l_1 penalizacija poznatija pod nazivom **LASSO** (least absolute shrinkage and selection operator).

$$\min_\theta \frac{1}{2m}||X\theta - y||_2^2 + \lambda||\theta||_1 \tag{2.17}$$

Ovaj algoritam je zanimljiv zato što često daje rješenje sa manje parametara pa se osim za regularizaciju može koristiti i za redukciju parametara.
Za regularizaciju možemo koristiti i elastic net algoritam koji kombinira l_1 i l_2 penalizaciju

$$\min_\theta \frac{1}{2m}||X\theta - y||_2^2 + \lambda_1\rho||\theta||_1 + \frac{\lambda_2(1-\rho)}{2}||\theta||_2^2 \quad 0 \leq \rho \leq 1. \tag{2.18}$$

3 Logistička regresija

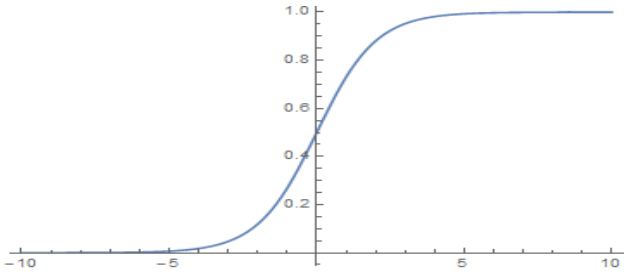
Unatoč imenu, logistička regresija je **klasifikacijski** model, ali se ovaj naziv koristi iz povjesnih razloga. Neki od problema na kojima se primjenjuje logistička regresija su spam filteri, online kupovina (koristi li netko ukradene podatke ili ne), procjena tumora (je li tumor zloćudan ili ne), vremenska prognoza...

3.1 Binarna klasifikacija

3.1.1 Funkcija procjene

Promotrimo prvo slučaj u kojem varijabla y koju želimo predvidjeti može poprimiti samo dvije vrijednosti, tj. $y \in \{0, 1\}$.
Klasu 0 zvat ćemo "negativna" klasa, a klasu 1 "pozitivna" klasa.
Za klasifikaciju ne možemo koristiti prijašnju definiciju funkcije $h_\theta(x)$, jer se za tako definiranu funkciju može dogoditi $h_\theta(x) \gg 1$ ili $h_\theta(x) \ll 0$ iako je u svim primjerima za učenje

$y \in \{0, 1\}$. Umjesto toga želimo pronaći funkciju h_θ td. je $\forall x, 0 \leq h_\theta(x) \leq 1$. Definirajmo prvo funkciju $\sigma(z) := \frac{1}{1+e^{-z}}$ i pogledajmo njezin graf.



Slika 3.1: Graf funkcije $\sigma(x)$

Očito je $0 \leq \sigma(z) \leq 1, (\forall z \in \mathbb{R})$ pa ćemo funkciju procjene za logističku regresiju definirati pomoću funkcije σ na sljedeći način

$$h_\theta(x) = \sigma(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{3.1}$$

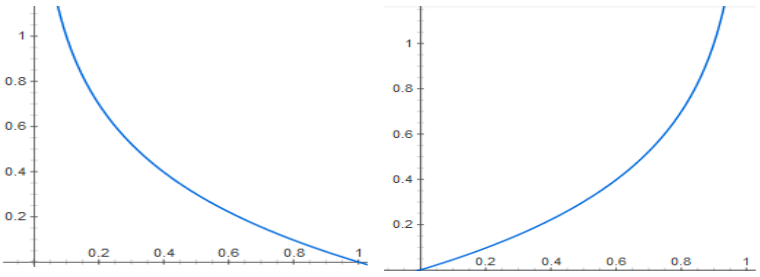
Vrijednost funkcije $h_\theta(x)$ ćemo interpretirati kao vjerojatnost da je $y = 1$ uz uvjet x parametrizirano sa θ ($h_\theta(x) = P(y = 1|x; \theta)$). Tada vjerojatnost da je $y = 0$ možemo izraziti kao $1 - h_\theta(x)$ ($P(y = 0|x; \theta) = 1 - P(y = 1|x; \theta)$).
Uočimo da je $h_\theta(x) \geq 0.5 \iff \theta^T x \geq 0$ pa će za klasifikaciju biti dovoljno provjeriti vrijednost izraza $\theta^T x$. Krivulju $\theta^T x$ nazivamo granica odlučivanja.

3.1.2 Funkcija cijene i MNS

Za logističku regresiju ne možmo koristiti ranije definiranu funkciju cijene (2.2), jer za ovako definiranu funkciju h_θ cijena više nije konveksna funkcija pa MNS nebi nužno pronašla globalni minimum. Radi jednostavnijeg zapisa prvo ćemo definirati funkciju

$$C(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{za } y = 1 \\ -log(1 - h_\theta(x)) & \text{za } y = 0 \end{cases}$$

Funkciju cijene sada možemo zapisati $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} C(h_\theta(x^{(i)}), y^{(i)})$. Razlog ovakve definicije jasno je vidljiv iz grafa funkcije C Možemo vidjeti da ako je $y = 1$ i $h_\theta(x) = 1$ C je 0.



(a) y=1 (b) y=0

Slika 3.2: Grafovi funkcije $C(h_\theta(x), y)$

Ako je $y = 1$, a naša funkcija predvidi vrijednost 0, tada $C \to \infty$.
Funkciju C možemo kompaktnije zapisati na sljedeći način

$$C(h_\theta(x), y) = -ylog(h_\theta(x)) - (1 - y)log(1 - h_\theta(x)) \tag{3.2}$$

Sada konačno možemo zapisati puni oblik funkcije cijene

$$J(\theta) = -\frac{1}{m}\Big[\sum_{i=1}^{m} y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))\Big] \tag{3.3}$$

Minimizaciju funkcije cijene ćemo, kao i kod linearne regresije, vršiti pomoću MNS.

$$\theta_j = \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{3.4}$$

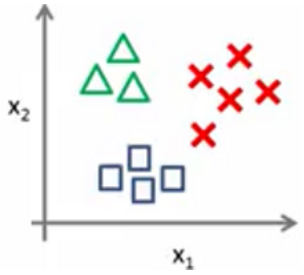
Prilikom implementacije treba paziti da se ovaj algoritam, zbog identičnog izgleda, ne zamijeni sa algoritmom koji smo definirali za linearnu regresiju. Razlika ova dva algoritma je u funkciji $h_\theta(x)$. Za linearnu regresiju imali smo $h_\theta(x) = \theta^T x$, dok je ovdje $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$. Kao i kod linearne regresije, transformacija podataka i regularizacija znatno će poboljšati brzinu i preciznost algoritma. Za regularizaciju logističke regresije najčešće se koristi l_2 penalizacija

$$\begin{aligned}
J(\theta) &= -\Big[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))\Big] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2 \\
\theta_0 &= \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)} \\
\theta_j &= \theta_j - \alpha\Big[\frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m}\theta_j\Big]
\end{aligned} \tag{3.5}$$

Transformaciju podataka provodimo na identičan način kao u slučaju linearne regresije.

3.2 Višestruka klasifikacija

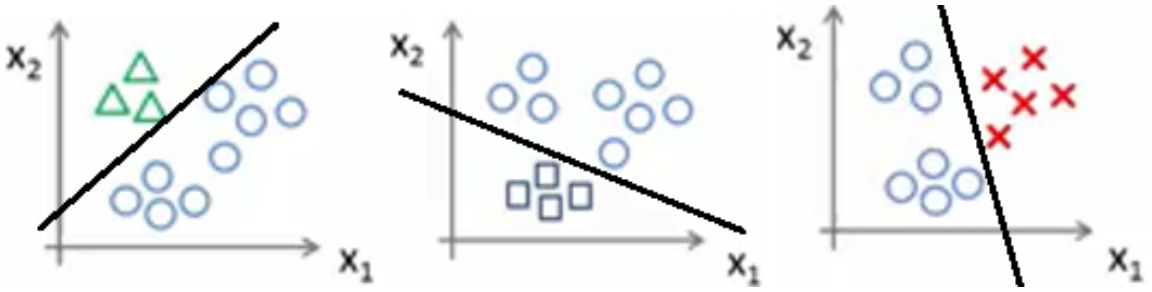
U slučaju da y može poprimiti više od dvije vrijednosti najjednostavniji "one-vs-rest"[3]. Cilj one-vs-rest algoritma je problem višestruke klasifikacije pretvoriti u problem binarne klasifikacije. Način rada ove metode ilustrirat ćemo na primjeru iz [1]



Slika 3.3: Klasifikacijski problem sa 3 klase

Za svaku od ove 3 klase konstruirat ćemo i riješiti zaseban binarnan klasifikacijski problem.

[3]Često se koristi skračenica "ovr"



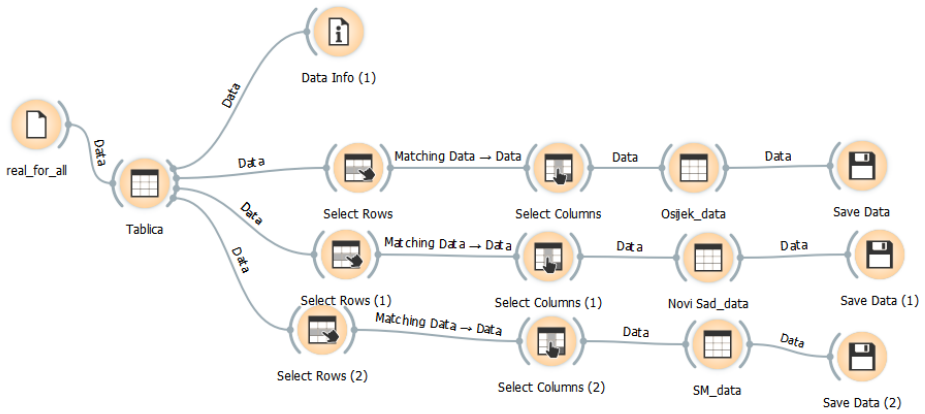
(a) y=1 $h_\theta^{(1)}(x)$ (b) y=2 $h_\theta^{(2)}(x)$ (c) y=3 $h_\theta^{(3)}(x)$

Slika 3.4: Riješenja binarnih klasifikacijskih problema

Za novi ulazni primjer predvijet ćemo klasu $i = \arg\max_i h_\theta^{(i)}(x)$.

4 Praktični projekt

U sklopu ovog završnog rada odrađen je i praktični projekt na podacima zavoda za javno zdravstvo. Cilj projekta bio je predvidjeti početak sezone ambrozije na temelju prikupljenih meteoroloških podataka i mjerenja koncentracije ambrozije u zraku. Za obradu prikupljenih podataka i kreiranje svojstava korišten je Orange, a zatim su za ta svojstva napravljeni modeli linearne regresije pomoću scikit-learn Python modula.



Slika 4.1: Orange workflow

```
enet = ElasticNet(alpha = 0.5,l1_ratio=0.5)
enet.fit(X_scaled,y)
print(enet.coef_)
```

Slika 4.2: Konstrukcija Elastic Net modela

Na temelju dobivenih koeficijenata možemo analizirati ovisnost početka sezone ambrozije o meteorološkim uvjetima. Dokumentaciju scikit-learn modula može se pronaći u [2] ili na `http://scikit-learn.org/stable/documentation.html`, a kod linearne regresije za ovaj projekt nalazi se u Prilogu 1.

Literatura

[1] A. Ng. Machine learning. https://www.coursera.org/learn/machine-learning.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Prilog 1

```
import csv
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
X = []
y = []
with open("data.csv") as file:
    r = csv.reader(file)
    for row in r:
        temp = row
        for i in range(len(temp)):
            temp[i] = float(row[i])
        X.append(temp)
X = np.array(X)
with open("target.csv") as file:
    r = csv.reader(file)
    for row in r:
        y.append(int(row[0]))
f = open("regresija.txt","w")
f.write("mnt6  mkt6   srt7   mkt7   mnt7   att7   vlz7   pad7   sbv7   mbv7\n")

lin_reg = LinearRegression()
lin_reg.fit(X_scaled,y)
f.write("Linearna regresija:\n")
for i in lin_reg.coef_:
    f.write(str(i)+" ")

lasso = Lasso(alpha = 0.5)
lasso.fit(X_scaled,y)
f.write("\nLASSO:\n")
for i in lasso.coef_:
    f.write(str(i)+" ")

enet = ElasticNet(alpha = 0.5,l1_ratio=0.5)
enet.fit(X_scaled,y)
f.write("\nElastic Net\n")
for i in enet.coef_:
    f.write(str(i)+" ")

ridge = Ridge(alpha=0.5)
ridge.fit(X_scaled,y)
f.write("\nRidge:\n")
for i in ridge.coef_:
    f.write(str(i)+" ")
f.close()
```