VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
MATEMATINĖS INFORMATIKOS KATEDRA

Kursinis darbas

Klasifikavimo metodų analizė

Atliko: 4 kursas, matematinė informatika
Ignas Jatulis (parašas)
Darbo vadovas:
lekt. Irus Grinis (parašas)

Vilnius
2018

Turinys

Įvadas

Šiais laikais žmonės vis daugiau veiksmų atlieka kompiuterio pagalba. Pavyzdžiui, jei seniau studentai sesijos metu, prieš egzaminus, konspektus rašydavo ranka, tai šiais laikais, vis daugiau studentų renkasi kompiuterį, jame esančias teksto tvarkykles, kurių pagalba galima patogiai ir tvarkingai parengti konspektus. Tačiau sudarydami konspektus iš ranką rašytų sąsiuvinių, jie sugaišta nemažai laiko. Tad kodėl nesukūrus kompiuterinės programos ar mobiliosios programėlės, kurios pagalba užtektų nufotografuoti ar nuskenuoti tekstą, o kompiuteris jį pats atpažintų?

Taigi, pratęsdamas Tiriamojo seminaro temą, šiame darbe išbandysiu pasirinktus vienus populiariausių raidžių atpažinimo metodus, palyginsiu gautus rezultatus bei tyrimo eigą. Gauti rezultatai pradės nuspręsti, kuris ar kurie metodai geriausiai tiktų, ar galima iš jų kažkurį naudoti mano bakalauro darbe, kurio galutinis tikslas - ranka rašyto teksto atpažinimo programėlė. Šiam tikslui pasiekti, pasitelksime vieną populiariausių kompiuterinės regos bibliotekų OpenCV ir joje įgyvendintus vaizdų apdorojimo algoritmus bei Python programavimo kalbai skirtą scikit-learn paketą, kuriame yra realizuoti populiariausi mašininio mokymo algoritmai.

1. Klasifikatoriai

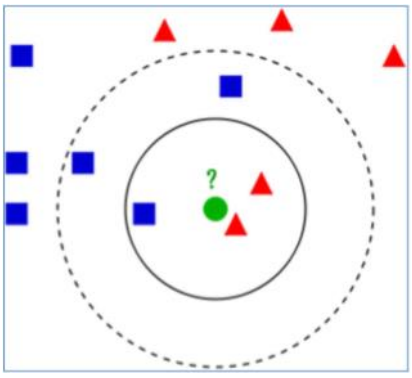
Klasifikavimą galime apibrėžti kaip duomenų paskirstymą tam tikrai grupei pagal požymius. Pavyzdžiui, atpažįstant raides, tai galime laikyti kaip nuotraukų segmentų priskyrimą tam tikroms reikšmių klasėms. Taip pat, šalia klasifikavimo, galime pamatyti kitą, labai panašų klasifikavimui metodą, tai yra klasterizavimą. Klasterizavimas – tai duomenų suskirstymas į tam tikras grupes pagal panašumą.

Šio darbo metu, klasifikavimą sudarys keli etapai. Pirmiausia, tai kiekvienas klasifikatorius bus apmokomas, kai bus pateikiami pradiniai duomenys su duomenis atitinkamomis reikšmėmis. Taip bus sudarytas klasifikatoriaus modelis. Sekančio etapo metu bus vykdomas klasifikatoriaus vertinimas, kuriuo metu bus skaičiuojamas klasifikavimo efektyvumas. Efektyvumas – tai dalis, kiek tiksliai bus grąžintų reikšmių iš visų nurodytų testinių duomenų. Neretai, kad būtų pasiekiami kuo geresni rezultatai, klasifikavimo metodai gali būti apjungiami, tačiau to šiame darbe neatlikau.

Toliau darbe panagrinėsime vienus iš populiariausių klasifikavimo metodų, kurie yra naudojami raidžių atpažinime.

1.1. K - artimiausių kaimynų (angl. KNN) klasifikatorius

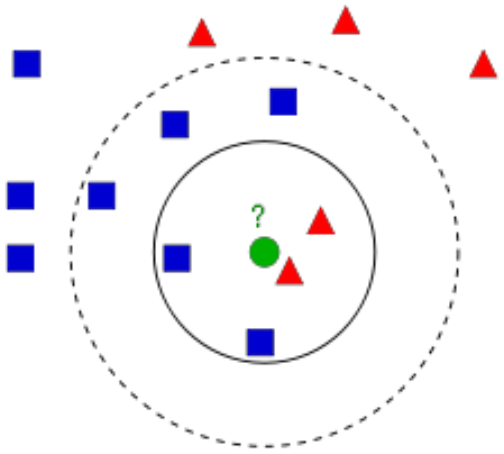
K - artimiausių kaimynų algoritmas (angl. K-Nearest Neighbors, KNN) – tai vienas iš paprasčiausių ir efektyviausių, neparametrinis klasifikavimo metodas ([5]). Algoritmo veikimo esmė, tai tam tikrų duomenų objektą priskirti konkrečiai klasei, priklausant nuo tą duomenų objektą supančių „kaimyninių“, tai yra panašių duomenų klasės reikšmių ir jų skaičiaus K. Duomenų objektas yra priskiriamas prie tos grupės duomenų, kurios duomenų objektų skaičius yra didžiausias nurodytame paieškos atstume. Toliau pateiksime pavyzdį (žr. pav. 1 iliustraciją), kaip veikia artimiausio kaimyno algoritmas.



pav. 1 Artimiausio kaimyno algortimo sprendžiama problema

Tarkime jog turime duomenų klases, kurios yra pažymėtos mėlynais kvadratais ir raudonais trikampiais. Šie duomenys yra kažkaip sukvalifikuotos. Paimkime kažkokį objektą, kurį pažymime kaip žalios spalvos apskritimą, kuris gali būti priskiriamas raudonos spalvos trikampių arba mėlynos spalvos stačiakampių klasei. Jeigu kaimynų skaičius K būtų 3, tada objektas, kurį pažymėjome kaip apskritimą, būtų priskiriamas raudonų trikampių klasei, kadangi 2 iš 3 kaimyninių objektų yra trikampiai, jų daugiausiai. Jeigu kaimynų skaičius K būtų 5, tada objektas, kurį pažymėjome kaip apskritimą būtų priskiriamas stačiakampių klasei, kadangi 3 iš 5 artimiausių kaimyninių objektų yra stačiakampiai.

Panagrinėkime dar vieną atvejį (žr. pav. 2). Situacija ta pati, turime duomenų klases, kurios yra pažymėtos mėlynais kvadratais ir raudonais trikampiais. Kaip ir prieš tai, paimkime kažkokį objektą, kurį pažymėsime žalius apskritimus ir kuris gali būti priskiriamas raudonos spalvos trikampių arba mėlynos spalvos stačiakampių klasei. Jei ieškomų kaimynų skaičius K būtų 4, tada susidurtume su situacija, kai tiek raudonų trikampių, tiek mėlynų stačiakampių „kaiminystėje“ turėsime po tiek pat, po du. Tada ir neaiškų, kuriai grupei priskirti mūsų objektą. Dėl to yra rekomenduojama imti nelyginį skaičių K kaimynų, kad išvengti situacijos kuri pavaizduota šiame pavyzdyje.



pav. 2 Lyginis kaimynu skaičius

Artimiausių kaimynų skaičius K visada yra nurodomas prieš pradedant klasifikavimo procesą. Algoritmo skaičiavimuose taip pat yra naudojami ir objekto atstumai iki kaimyninių elementų. Atstumas gali būti apskaičiuojamas skirtingais būdais, pavyzdžiui: Euklido atstumu (angl. Euclidean distance), Minkowski atstumu (angl. Minkowski distance), Hamingo atstumu (angl. Hamming distance) ir kitais matematiniais atstumų matavimo būdais.

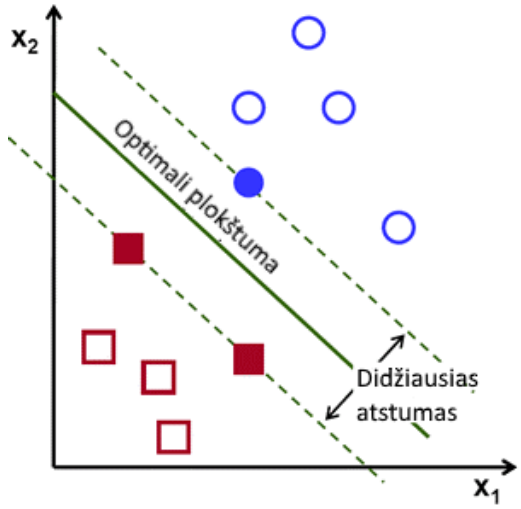
Kaip ir kiekvienas algoritmas, K-artimiausių kaimynų algoritmas turi jų. Vienas didžiausių, tai kad algoritmas yra priklausomas nuo to, kaip yra pasiskirstę duomenys pagal kiekį, t.y. jeigu viena

klasė turės daug daugiau duomenų nei kitos klasės, tai tokiu atveju, klasifikatorius bus parankesnis didžiausiai duomenų klasei. Tai bus dėl to, kad tiesiog didžiosios klasės objektų visada bus daugiau „kaimynystėje“. Tam ištaisyti yra naudojami svorio ryšio koeficientai kiekvienai iš klasių bei skaičiuojami atstumai iki objekto kaimyninių elementų ([7]).

1.2. Atraminių vektorių mašinos (angl. SVM) klasifikatorius

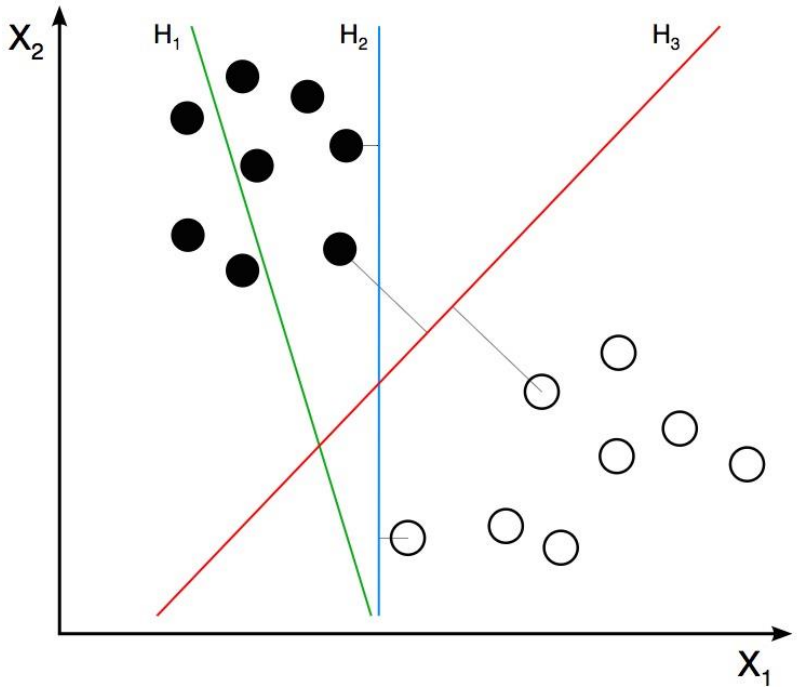
Atraminių vektorių mašinų klasifikatoriaus (angl. support vector machine, SVM) buvo sugalvotas Vladimiro Vapniko ir paskelbtas 1995 metais. Nors algoritmas yra ganėtinai naujas, tačiau jis yra plačiai taikomas įvairių objektų atpažinimo, klasifikavimo problemų sprendimui įvairiose srityse, tokiose, kaip dokumentų klasifikavimas, kompiuterinė rega ir kt.

Paanalizuokime kaip veikia algoritmas pasitelkdami pavyzdį (žr. pav 3).



pav. 3 Atraminių vektorių mašinų klasifikatoriaus veikimas

Atraminių vektorių mašinų klasifikatoriaus idėja yra surasti optimalią plokštumą, daugiamatiškumo atveju - hiperplokštumą (angl. Hyperplane), tokią, kuri skirtų dvi klases. Tarkime turime dvi klases duomenų, kvadratus ir apsikritimus. Ieškome vektorių, kurie atskirtų duomenų grupes. Mūsų ieškoma plokštuma turi skirti duomenų klases kiek įmanoma didesniu atstumu. Ne visi vektoriai bus mums tinkami, pavyzdžiui, kaip pavaizduota paveikslėlyje žemiau (žr. pav. 4). Tarkime nustatome 3 vektorius. H1 yra netinkamas, nes neatskiria duomenų klasių, H2 ir H3 atskirai pusiau, tačiau H3 atskiria duomenų klases didžiausiu atstumu. Kai tokia plokštuma randama, galima klasifikuoti duomenis į dvi atskiras klases.



pav. 4 Vektorių paieškos pavyzdys

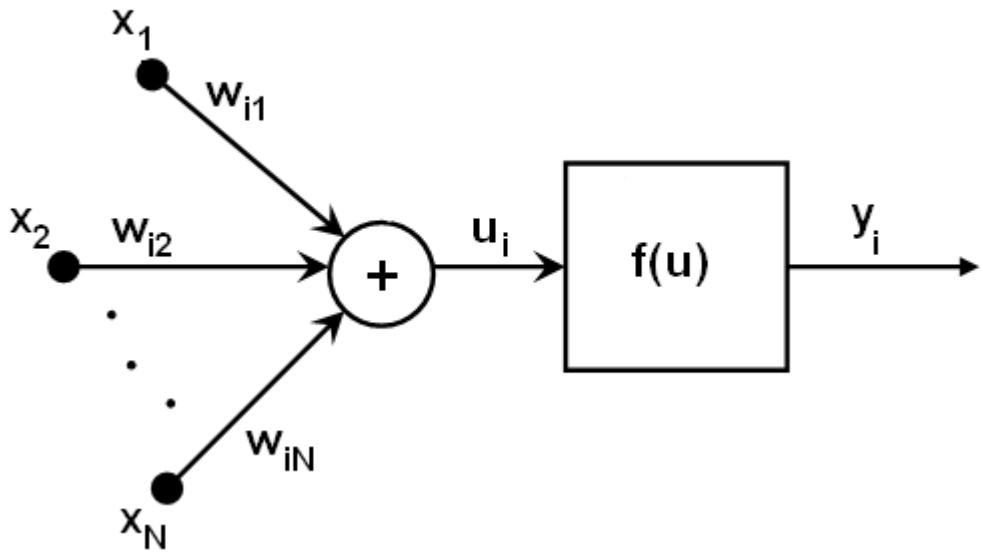
Kad apskaičiuoti optimaliausią plokštumą, jų skaičiavimui gali būti taikoma viena iš atraminių vektorių mašinų branduolio (angl. kernel) funkcijų. Taikomų branduolio funkcijų pavyzdžiais gali būti: tiesinis branduolys (angl. Linear kernel), sigmoidinis branduolys (angl. Sigmoid kernel), polinominis branduolys (angl. Polynomial kernel). Šių funkcijų ir parametrų paskirtis yra sutvarkyti gaunamus duomenis į daugiamatę erdvę pagal požymius, kurioje požymių klasės atskiriamos hiperplokštumomis daugiamatėje erdvėje.

1.3. Daugiasluoksnio parceptrono (angl. MLP) klasifikatorius

Daugiasluoksnio parceptrono (angl. Multilayer perceptron, MLP) klasifikatorius yra viena iš Dirbtinio neuroninio tinklo ANN (angl. Artificial Neural Network) klasių. Algoritmas yra vienas iš dažniausiai naudojamų algoritmų atpažįstant vaizdus. Šis klasifikavimo metodas naudoja idėją, kaip sąveikauja ir funkcionuoja neuronai esantys žmogaus smegenyse. Vienas neuronas, neuronų tinkle vienu metu gali susijungti su tūkstančiais kitų neuronų, kurie gali generuoti bei dalintis nauja informacija tarpusavyje. Neuronai yra susijungę su kitais neuronais tinkle per jungtis, kurios vadinamos sinapsėmis (angl. synapse).

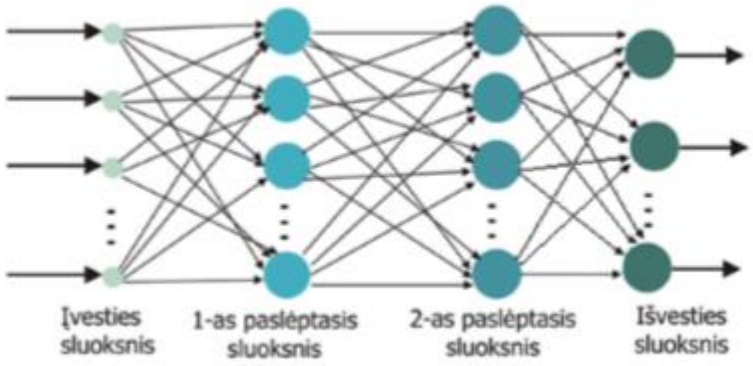
Parceptronas yra matematinis neurono modelis. Jo grafinė struktūra pavaizduota žemiau (žr. pav.5). Įėjimo vektoius X (x_1, x_2, ... x_N) yra dauginamas iš atitinkamo W(xi_1, xi_2, ... xi_N) svorių vektoriaus. Visos sandaugos yra sumuojamos, gaunamas U_i signalas, kuris perduodant aktyvavimo funkcijai f(u), gaunamas išėjimo signalas Y_i.

Kaip aktyvavimo funkcija dažniausiai yra naudojama slenkstinė funkciją, sigmoidinė funkcija, hiperbolinis tangentas ir kt.



pav. 5 Parceptrono pavyzdys

Paprastai MLP modelis susideda iš įėjimų sluoksnio, išėjimų sluoksnio ir vieno arba kelių paslėptų sluoksnių. Paslėpti sluoksniai padeda nustatyti, koks rezultatas turi būti. Per įėjimo sluoksnį patekę duomenys yra perduodami į išėjimo sluoksnį per paslėptus sluoksnius, kur kiekvienas neuronas yra sujungtas su sekančiu (žr. pav. 6).



pav. 6 Neuroninis tinklas su paslėptais sluoksniais

Kiekvienas sluoksnis susideda iš tam tikro skaičiaus neuronų ir kiekvienas neuronas turi svorinius koeficientus, kurie reikalingi tolimesniam apdorojimui. Norint rasti neuronų svorinius koeficientus, reikia atlikti neuronų mokymą. Neurono mokymo metu į modelį paduodamas mokymo rinkinys, kurį sudaro įėjimų ir tikslų reikšmės. Tada yra skaičiuojama nuostolių funkcija ir siekiama ją minimizuoti [8].

2. Programavimo aplinka ir įrankiai

Kursinio darbo metu, analizavimui skirta programinė įranga buvo kurta pasitelkiant Python programavimo kalbą, panaudojant OpenCV ir scikit-learn bibliotekas bei jose realizuotus algoritmus, kurie padeda apdoroti gautus vaizdus bei sukurti mūsų analizuojamus algortimus.

2.1. OpenCV biblioteka

Nuotraukų apdorojimui buvo pasirinkta OpenCV biblioteka ([2]). OpenCV (Open Source Computer Vision Library) yra atvirojo kodo kompiuterinės regos biblioteka. Šią biblioteką galima naudoti tiek mokslo, tiek komercijos tikslais. „OpenCV“ pradėta kurti vieno iš „Intel“ bendrovės padalinio 1999 metais. Šiuo metu „OpenCV“ labiausiai plėtoja ir palaiko kompiuterinę regą tyrinėjanti bendrovė „Itseez“. Kadangi biblioteka yra atvirojo kodo, jos išeities kodą galime surasti GitHub.com([1]) saugykloje, todėl prie jos tobulinimo ir kūrimo gali prisidėti kiekvienas.

„OpenCV“ kompiuterinės regos bibliotekoje yra realizuota daugiau kaip 2500 optimizuotų algoritmų kurie padeda apdirbti vaizdus. Originaliai ši biblioteka buvo parašyta su C++ kalba, tačiau ji taip pat turi sąsajas tokioms populiarioms programavimo kaip Python, JAVA ar MATLAB programiniai įrangai. Taip pat, ji yra suderinama su populiariausiomis operacinėmis sistemomis, tokiomis kaip Windows, Linux, Android ar Mac OS. Apie kiekvieną bibliotekoje esantį algoritmai ir jo panaudojimo pavyzdžius, galima rasti OpenCV dokumentacijoje ([1]).

2.2. scikit-learn biblioteka

Klasifikavimo algoritmų realizacijai buvo pasirinkta scikit-learn biblioteka. Scikit-learn (arba scikits.learn) yra nemokama biblioteka skirta Python programavimo kalbai. Joje yra realizuota daug algaritmų skirtų klasifikavimui, klasterizavimui ar regresijos skaičiavimui. Pirma bibliotekos versija buvo išleista 2007 metais, jos autorius David Cournapeau. Biblioteka yra parašyta naudojant Python, Cython ir C programavimo kalbas. Ją palaiko tokios operacinės sistemos kaip Linux, macOS ir Windows [10].

2.3. Programavimo aplinka ir priemonės

Klasifikavimo metodų tyrimui atlikti buvo naudojamas ASUS X550LB nešiojamasis kompiuteris su Intel® Core™ I7-4500U dviejų branduolių 1.80 GHz procesoriumi. Kompiuteryje yra naudojama 8 GB dydžio operatyvioji atmintis. Taip pat bandymų aplinkoje buvo įdiegta Windows 10 operacinė sistema. Eksperimentų metu buvo ruošiamas pradinių duomenų rinkinys papildomai apdoruojant nuotraukose esančius vaizdus (plačiau parašyta skyriuje, apie pradinių

duomenų šaltinį), tyrimo metodo architektūros duomenų užpildymas, lyginami metodų pritaikymo rezultatai.

3. Pradinių duomenų šaltinis / struktūra

Prieš klasifikavimo metodų palyginimo tyrimą, buvo ieškoma raidžių ir skaičių duomenų rinkinių. Yra begalė, nemokamai prieinamų duomenų bazių, kurios gali būti pritaikomos algoritmų apmokymui ir testavimui. Buvo pasirinkta vienos iš dažniausiu naudojamų „The Chars74K dataset“ ([6]). duomenų bazės dalis. Rinkinyje galima rasti 18 265 pavyzdžius, įvairiais kompiuterio šifrais parašytų skaitmenų nuo 0 iki 9 ir didžiųjų bei mažųjų angliškos abecėlės radžių klases. Kiekviena klasė turi apie 300 paveikslėlių, paveikslėliai yra 128 x 128 pikselių dydžio. Paveikslėlio pavyzdys pateiktas žemiau (žr. pav. 7).

pav. 7 Paveikslėlio pavyzdys

Raidžių pradinių ir testinių duomenų failai buvo papildomai apdoroti. Pirmiausia buvo vykdomas iškirpimas, t.y., buvo skaitomi stulpeliai ir eilutės bei nustatomos vietos, nuo kur iki kur yra simbolis nuotraukoje. Tai nustačius, paveikslėlis buvo apkarpomas ir galiausiai pakeistas jo dydis į 24x24 pikselių dydį. Pertvarkytas paveikslėlis buvo išsaugomas tokiu formatu: *klasėspavadinimas.(eilėsNr).bmp*. Pertvarkyta pradinių duomenų bazė bus naudojama klasifikavimo metodų analizėje.

Kadangi surastoje duomenų bazėje paveiksliukai yra be foninio triukšmo, pasidarė įdomu, kokiu tikslumu pavyktų šiem algortimam suklasifikuoti vaizdus, kai jų fone kažkas dar yra. Tam tikslui patikrinti, nutariau pasirinkti klasikinį gėlių klasifikavimo ir atpažinimo uždavinį. Šiam patikrinimui atlikti, reikiamą duomenų rinkinį susiradau Kaggle.com [9] svetainėje. Šioje svetainėje kiekvienas žmogus gali pasidalinti savo surinktu duomenų rinkiniu, realizuoti atpažinimo algoritmus ir palyginti rezultatus su kitais svetainės naudotojais. Surastą duomenų rinkinį sudaro 5 klasių, t.y. nuotraukos, kuriose pavaizduota ramunėlių, tulpių, rožių, saulėgrąžų, kiaulpienių žiedai, kiekvieną klasę sudaro 734 nuotraukos, bendras duomenų rinkinio kiekis užima 215MB.

4. Rezultatų analizė

Išbandant metodus, buvo nuspręsta, kad analizuosime, kaip pasikeis rezultatai, jei bus keičiami šie klasifikatorių parametrai:

Klasifikatorius	**Parametras, kuris keičiamas**	**Keitimo ribos**
K – artimiausių kaimynų	Kaimynų skaičius	Nuo 1 iki 10, žingsnis kas 1
Atraminių vektorius mašina	Polinomo laipsnis	Nuo 1 iki 4, žingsnis kas 1
Daugiasluoksnis perceptronas	Paslėptų sluoksnių skaičius	Nuo 100 iki 1000, žingsnis kas 100

4.1. Raidžių duomenų rinkinio analizė

Prieš pradedant analizuoti klasifikatorius naudojant raidžių duomenų rinkinį, buvo nuskaitomi paveiksliukai. Nuskaityti duomenys buvo padalinti taip, kad 75 procentus sudarytų apmokymui skirti duomenys, o 25 procentai būtų skirti testavimui, t.y. nustatymui, kiek procentų pavyks teisiklingai priskirti duomenis reikiamai klasei (žr. pav. 8).

```
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import numpy as np
import imutils
import cv2
import os

def flatten_image(img):
    return img.flatten()

print("Nuskaitom nuotraukas")
imagePaths = list(paths.list_images("fnt"))

features = []
labels = []

for (i, imagePath) in enumerate(imagePaths):
    image = cv2.imread(imagePath)
    label = imagePath.split(os.path.sep)[-1].split(".")[0]
    hist = flatten_image(image)
    features.append(hist)
    labels.append(label)

le = LabelEncoder()
labels = le.fit_transform(labels)

print "Padalinam nuskaitytas nuotraukas, kad 75% sudarytu apmokymui, o 25% testavimui"
trainData, testData, trainLabels, testLabels = train_test_split(features, labels, test_size=0.25, random_state=42)
```

pav. 8 Paveikslėlių nuskaitymas

Tiriant K - artimiausių kaimynų rezultatų pokyti, kaip pasikeičia rezultatai nuo nurodytos K reikšmės, buvo naudojamas kodas:

```
for k in range(1, 11):
    print "K = %d" % (k)
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(trainData, trainLabels)
    predictions = model.predict(testData)
    print(classification_report(testLabels, predictions, target_names=le.classes_))
```

pav. 9 kNN realizacijos kodas

Gautų rezultatų lentelė pateikiama žemiau. Pirmoje eilutėje išvardintos K koeficiento reikšmės, o antroje eilutėje, dalis kiek testinių duomenų buvo teisingai suklasifikuota.

K	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Dalis	0.92	0.88	0.87	0.87	0.86	0.86	0.86	0.86	0.85	0.85

Tyrimo metu gauti rezultatai parodo, kad dydžiausią tikslumą pavyko pasiekti tada, kai buvo ieškoma vos vieno kaimyno erdvėje. Kai K koeficientas buvo 1, buvo gautas 92 proc tikslumas.

Tiriant Atraminių vektorių mašinos klasifikatoriaus metodo rezultatus, buvo pasirinkta, jog bus keičiamas polinomo laipsnis.

```
for p in range(1, 5):
    print "Polinomo laipsnis = %d" % (p)
    model = SVC(kernel='poly', degree=p, gamma=1)
    model.fit(trainData, trainLabels)
    predictions = model.predict(testData)
    print(classification_report(testLabels, predictions, target_names=le.classes_))
```

pav. 10 SVM realizacijos kodas

Kaip keitėsi rezultatų atpažinimo tikslumas pateikiama žemiau. Pirmoje eilutėje nurodomas polinomo laipsnis, o antroje eilutėje, dalis kiek testinių duomenų buvo teisingai suklasifikuota.

Polinomo laipsnis	**1**	**2**	**3**	**4**
Dalis	0.91	0.91	0.90	0.90

Tyrimo metu gauti rezultatai parodo, kad dydžiausią tikslumą pavyko pasiekti tada, kai polinomas buvo pirmo arba antro laipsnio. Su šių laipsnių polinomais, buvo gautas 91 proc tikslumas.

Tiriant Daugiasluoksnio parceptrono klasifikatorių, buvo tiriama priklausomybe nuo jame esančio paslėptų sluoksnių skaičiaus.

```
for l in range(1, 11):
    print "Sluoksniu skaicius = %d" % (l*100)
    model = MLPClassifier(hidden_layer_sizes=(l*100, ))
    model.fit(trainData, trainLabels)
    predictions = model.predict(testData)
    print(classification_report(testLabels, predictions, target_names=le.classes_))
```

pav. 11 MLP klasifiaktoriaus realizacijos kodas

Kaip keitėsi rezultatų priskyrimo tikslumas, pateikiama žemiau esančioje lentelėje. Pirmoje eilutėje nurodomas paslėptų sluoksnių skaičius, o antroje eilutėje, dalis kiek testinių duomenų buvo teisingai suklasifikuota:

Paslėptų sluoksnių skaičius	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
Dalis	0.14	0.35	0.76	0.78	0.79	0.79	0.82	0.82	0.80	0.81

Tyrimo metu gauti rezultatai parodo, kad dydžiausią tikslumą pavyko pasiekti tada, kai paslėptų sluoksnių skaičius buvo tarp 700 ir 800. Su tokiu skaičiumi sluoksnių, buvo gautas 82 proc tikslumas.

4.2. Gėlių duomenų rinkinio analizė

Kaip ir su raidžių duomenų rinkiniu, taip ir su gėlių nuotraukų rinkiniu atliksime tuos pačius patikrinimus. Pažiūrėsime kaip keisis rezultatų tikslumas, keičiant tam tikrą parametrą mūsų pasirinktuose klasifikavimo algoritmuose.

```
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import numpy as np
import imutils
import cv2
import os

def extract_color_histogram(image, bins=(8, 8, 8)):
    # extract a 3D color histogram from the HSV color space using the supplied number of `bins` per channel
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist = cv2.calcHist([hsv], [0, 1, 2], None, bins, [0, 256, 0, 256, 0, 256])
    cv2.normalize(hist, hist)
    return hist.flatten()

print("Nuskaitom nuotraukas")
imagePaths = list(paths.list_images("flowers_dataset"))

features = []
labels = []

for (i, imagePath) in enumerate(imagePaths):
    image = cv2.imread(imagePath)
    label = imagePath.split(os.path.sep)[-1].split(".")[0]
    hist = extract_color_histogram(image)
    features.append(hist)
    labels.append(label)

le = LabelEncoder()
labels = le.fit_transform(labels)

print "Padalinam nuskaitytas nuotraukas, kad 75% sudarytu apmokymui, o 25% testavimui"
trainData, testData, trainLabels, testLabels = train_test_split(features, labels, test_size=0.25, random_state=42)
```

pav. 12 Gėlių duomenų rinkinio nuskaitymas

Kadangi algoritmų realizaciją yra tokia pat kaip ir prieš tai naudotoje analizėje, toliau pateiksiu tirimų rezultatus.

K – artimiausių kaimynų algoritmas geriausiu atveju pavyko teisiklingai suklasifikuoti 46 procentus duomenų. Toks rezultatas pasiektas, kai parametras K buvo tarp 4 ir 7 bei 10.

K	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Dalis	0.42	0.43	0.44	0.46	0.46	0.46	0.46	0.46	0.45	0.46

Atraminių vektorių mašinų klasifikatorius geriausiu atveju teisingai suklasifikavo 56 procentus duomenų. Tai pasiekta, kai buvo naudojamas antro laipsnio polinomas.

Polinomo laipsnis	**1**	**2**	**3**	**4**
Dalis	0.54	0.56	0.55	0.54

Geriausias rezultatas buvo pasiektas tiriant Daugiasluoksnio parceptrono klasifikatorių, kai jis buvo sudarytas iš 400 paslėptų sluoksnių. Jo metu, buvo pasiektas 59 procentų tikslumas, kas yra geriausias tarp mūsų pasirinktų algoritmų.

Paslėptų sluoksnių skaičius	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
Dalis	0.59	0.56	0.58	0.59	0.57	0.57	0.58	0.59	0.58	0.57

Išvados

Šiame darbe buvo nustatyta, jog pasitelkus K - artimiausių kaimynų algoritmą, buvo teisingai atpažinta 92 procentai raidžių ir skaitmenų. Rezultatas buvo geriausias, kai artimiausių kaimynų koeficientas K buvo pasirinktas 1. Šis klasifikavimo metodas buvo tiksliausias, palyginus su Atraminių vektorių mašinų metodu, kurio geriausiu bandymu pavyko užfiksuoti 91 procentą bei Daugiasluoksniu parceptronu, kurio geriausias rezultatas buvo 82 procentai.

Nors geriausius rezultatus pavyko pasiekti su K – artimiausių kaimynų algoritmu, tačiau tam didelę įtaką turėjo pasirinktas pradinių duomenų bazė. Ten paveiksliukai buvo be „triukšmo“ tad mano manymu, jis labai tik šiam klasifikavimo algoritmui.

Šio darbo metu susipažinau su ne tik šiais trimis kalsifikavimo algoritmais, bet ir kitais mašininio mokymo algortimais, kurie naudojami vaizdų atpažinimui. Pagal perskaitytus straipsnius ir ką parodė antras tyrimas, kuriame buvo naudojamas duomenų rinkinys su „triukšmu“, pastebėjau, jog geriausi rezultatai raidžių atpažinime, ką aš bandysiu padaryti bakalaurinio darbo metu, pasiekiami naudojant dirbtinius neuroninius tinklus. Vienas tokių yra sąsukiniai arba kitaip vadinami konvoliuciniai neuroniniai tinklai (angl. Convolutional neural networks, ConvNets, CNN), kurių pagalba bandysiu įgyvendinti savo tolimesnį, bakalauro darbą.

Literatūra

[1] OpenCV documentacija. http://docs.opencv.org/ [Žiūrėta 2018 m. sausio mėn.]

[2] Apie OpenCV. http://opencv.org/about.html [Žiūrėta 2018 m. sausio mėn.]

[3] G. Kavaliauskas, G. Felinskas. Dirbtinio intelekto atpažinimo metodų analizė ir taikymai ranka rašytam tekstui atpažinti. Jaunųjų mokslininkų darbai (Nr. 4(37)), 2012, p. 205-211

[4] H.S.M. Beigi. An overview of handwriting recognition. Proceedings of the 1 st Annual Conference on Technological Advancements in Developing Countries, 1993, p. 30-46

[5] Gao, Y.; Li, C.; Chen, G.; Chen, L.; Jiang, X.; Chen, C. 2007. Efficient k-Nearest-Neighbour Search Algorithms for Historical Moving Object Trajectories, Journal of Computer Science and Technology 22(2): 232–244

[6] http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/ [Žiūrėta 2018 m. sausio mėn.]

[7] G. Žukas. Kalbos emocijų požymių tyrimas. Magistro baigiamasis darbas, 2014

[8] A. Andrejevas. Rašto ženklų atpažinimas naudojant neuroninius tinklus. 2011

[9] https://www.kaggle.com/alxmamaev/flowers-recognition [Žiūrėta 2018 m. sausio mėn.]

[10] http://scikit-learn.org/stable/documentation.html [Žiūrėta 2018 m. sausio mėn.]