

Izbor instanci podataka (Instance Selection)

**Prikupljanje i predobrada
podataka za mašinsko učenje**

Mentor:

Prof. dr Aleksandar S. Stanimirović

Student:

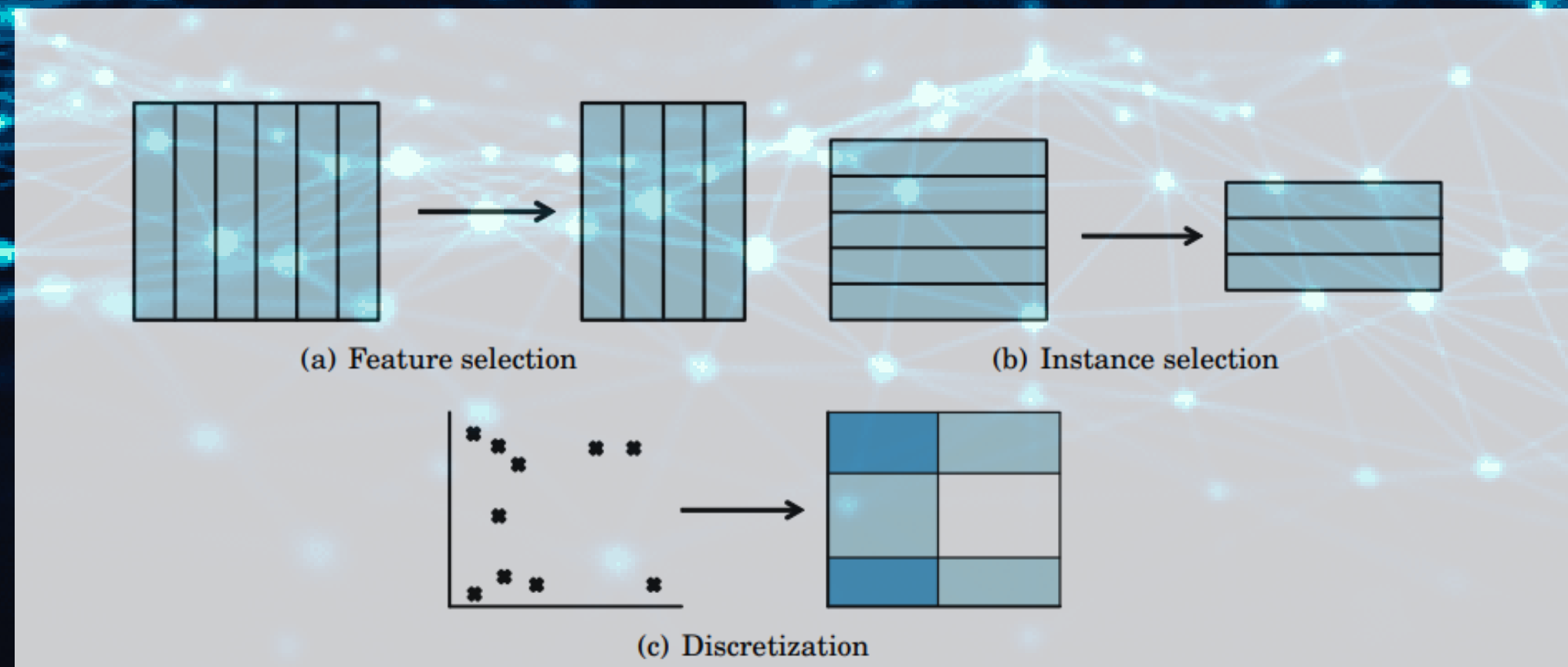
Petrović Nikola 1466

Sadržaj

- Redukcija dimenzionalnosti
- Izbor instanci podataka (Instance selection)
- Prototype selection (PS) - Osnovni principi
- Algoritmi PS
- Praktična primena algoritama
- Zaključak

Redukcija dimenzionalnosti

- Diskretizacija (eng. Discretization)
- Ekstrakcija karakteristika (eng. Feature extraction)
- Generisanje instance (eng. Instance generation)
- Izbor atributa (eng. Feature selection)
- Izbor instanci podataka (eng. Instance selection)



Instance selection

Definicija?

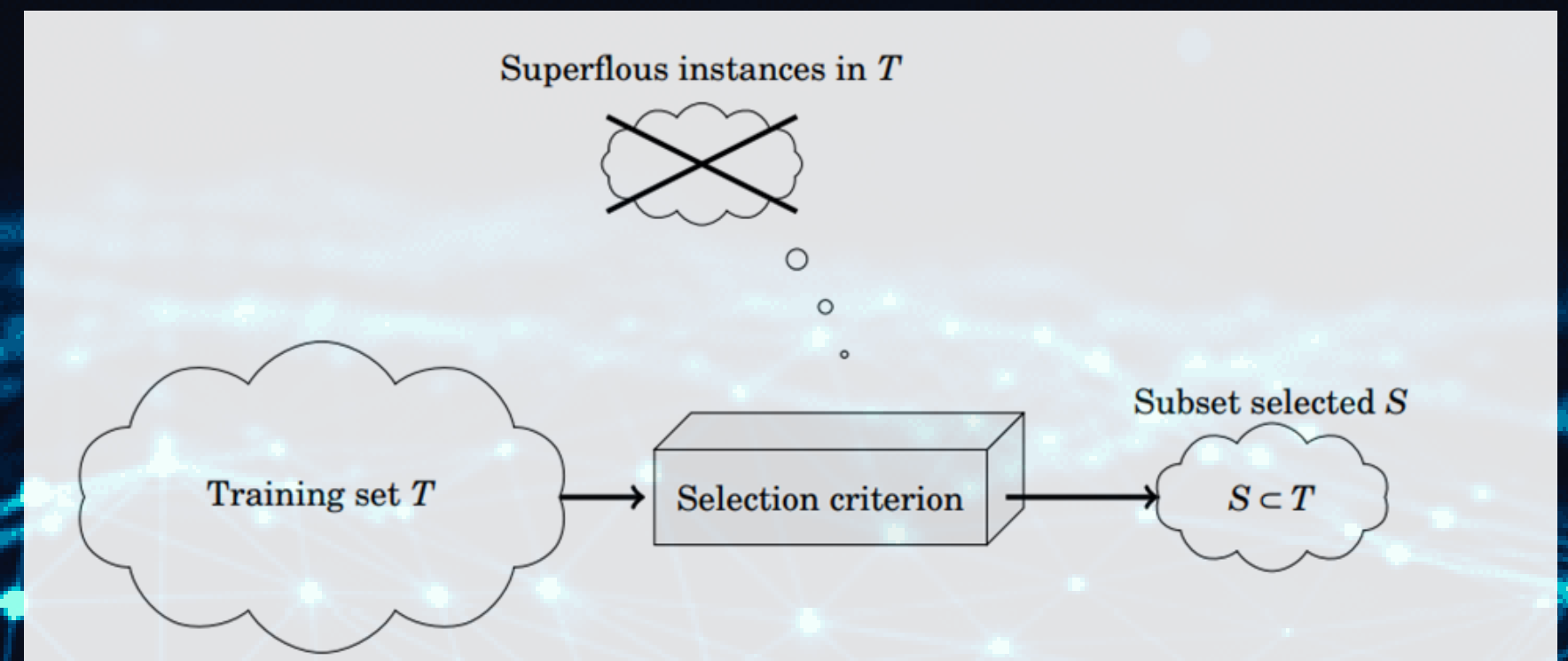
- Proces odabira najreprezentativnijih primera iz skupa podataka.

Ocekivan rezultat?

- Minimalni podskup podataka kompletnog skupa, nezavistan od modela, koji može obavljati isti zadatak bez gubitka performansi.

Osnovni zadaci:

1. Optimizacija korišćenja memorijskih resursa
2. Smanjenje vremena obrade skupa podataka
3. Uklanjanje šuma i suvišnih instanci



PS (Prototype Selection)

Definicija?

- PS metode su IS metode koje očekuju da pronađu skupove za treniranje koji pružaju najbolju tačnost klasifikacije i stopu smanjenja korišćenjem klasifikatora zasnovanih na instanci koji uzimaju u obzir određenu sličnost ili meru udaljenosti.

Osnovne karakteristike:

- Efikasniji pristup od TSS (eng. Training Set Selection)
- Omogućavanje brzog vremena izvršavanja
- Minimalni gubici tačnosti klasifikacije
- Osetljivost na redosled čitanja podataka, outlier-e i bučne podatke

Podela PS metoda po smeru pretrage

Inkrementalni (eng. Incremental)

- Počinje se praznim skupom S , dodaju se instance iz TR koje zadovoljavaju određene kriterijume u S
- Zavise od redosleda čitanja podataka
- Slučajan odabir
- Instance se mogu naknadno dodavati u skup S
- Minimalni zahtevi za memorijom tokom obrade podataka
- Skloni greškama

Dekrementalni (eng. Decremental)

- Počinje se sa $S = TR$, zatim se traže instance koje su suvišne
- Bitan redosled čitanja podataka
- Vremenski skupa operacija
- Faza učenja mora da se izvede van mreže

Grupni (eng. Batch)

- Za svaku instancu se proveravada li zadovoljava određene kriterijume pre uklanjanja bilo koje od njih
- Sve instance se uklanjaju istovremeno
- Vremenski skup metod

Mesoviti (eng. Mixed)

- Nadogradnja na Incremental i Decremental metode, nakon odabira skupa S , mogu se dodati ili ukloniti instance na osnovu određenih kriterijuma
- Poboljšava preciznost
- Vremenski zahtevne metode

Fiksni (eng. Fixed)

- Konačan broj instanci podataka je unapred određen

Tipovi selekcije kod PS metoda

Kondenzovanje (eng. Condensation)

- Ove tehnike imaju za cilj da zadrže instance koje su bliže graničnim tačkama.
- Uklanjanje većeg dela instanci
- Dobri rezultati nad trening skupom podataka, losi nad test skupom podataka


Izdanje (eng. Edition)

- Uklanjanje graničnih tačaka
- Minimalna redukcija
- Pобољшanje tačnosti predikcije

Hibridni pristup (eng. Hybrid)

- Za cilj imaju da poboljšaju tačnost predikcije i pronadju minimalni podskup S
- Uklanjanje unutrasnjih i graničnih tačaka
- Kombinacija prethodna dva pristupa

Kriterijumi za poredjenje PS metoda

- Smanjenje zauzeća memorijskog prostora (eng. Storage reduction)
 - Tolerancija buke (eng. Noise tolerance)
 - Tačnost generalizacije (eng. Generalization accuracy)
 - Vremenska zahtevnost (eng. Time requirements)
- 

PS Methods

Condensation

Edition

Hybrid

Incremental

Decremental

Batch

Decremental

Batch

Incremental

Decremental

Batch

Mixed + Wrapper

Fixed + Wrapper

CNN
Ullmann
TCNN
MNV
MCNN
GCNN
FCNN
PSC

RNN
SNN
Shrink
MCS
MSS

IKNN
POP
Reconsistent
TRKNN

ENN
Multiedit
RNGE
MENN
NCNEdit
ENRBF
ENNTh

AIKNN
MoCS

IB3

Filter
VSM
PF
DROP3
PSRCG
Cpruner
SVBPS
NRMCS

Wrapper
BSE

ICF
HMNEI
CCIS

Explore
GGA
CerveronTS
EDA
IGA
ZhangTS
CHC
GGA-MSE-
CC-FSM
SSMA

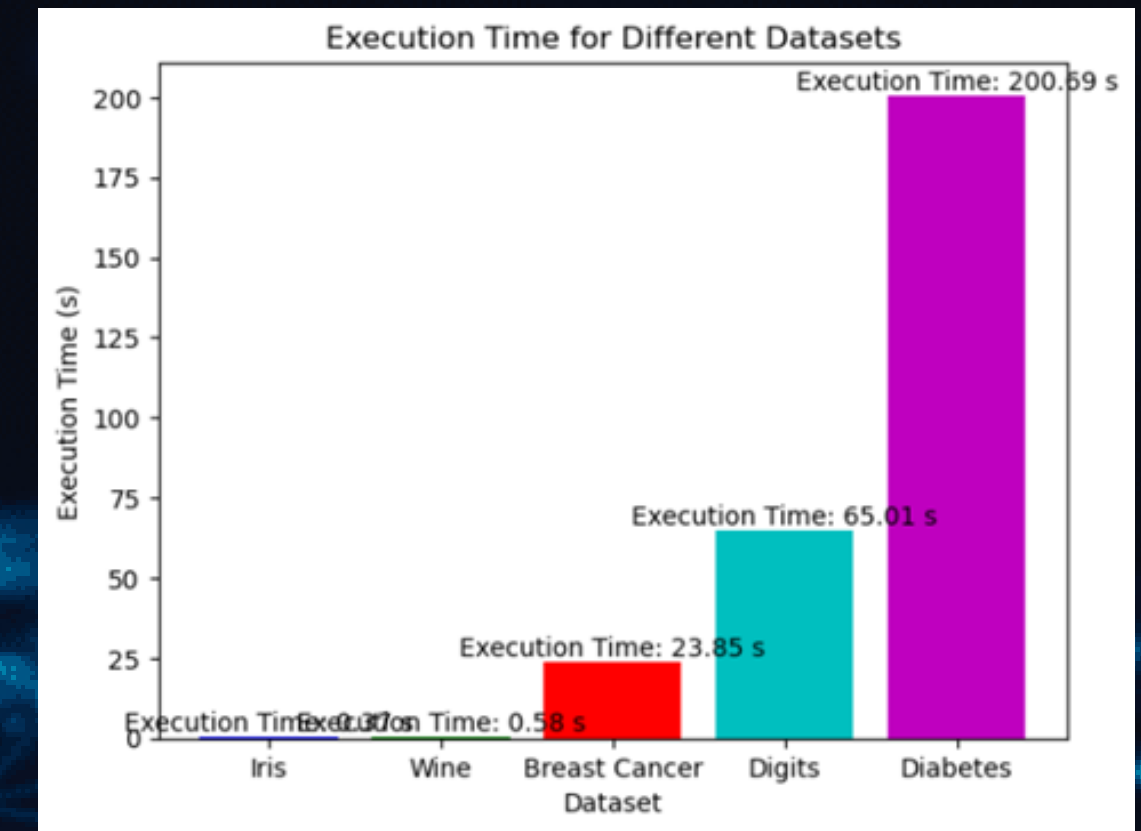
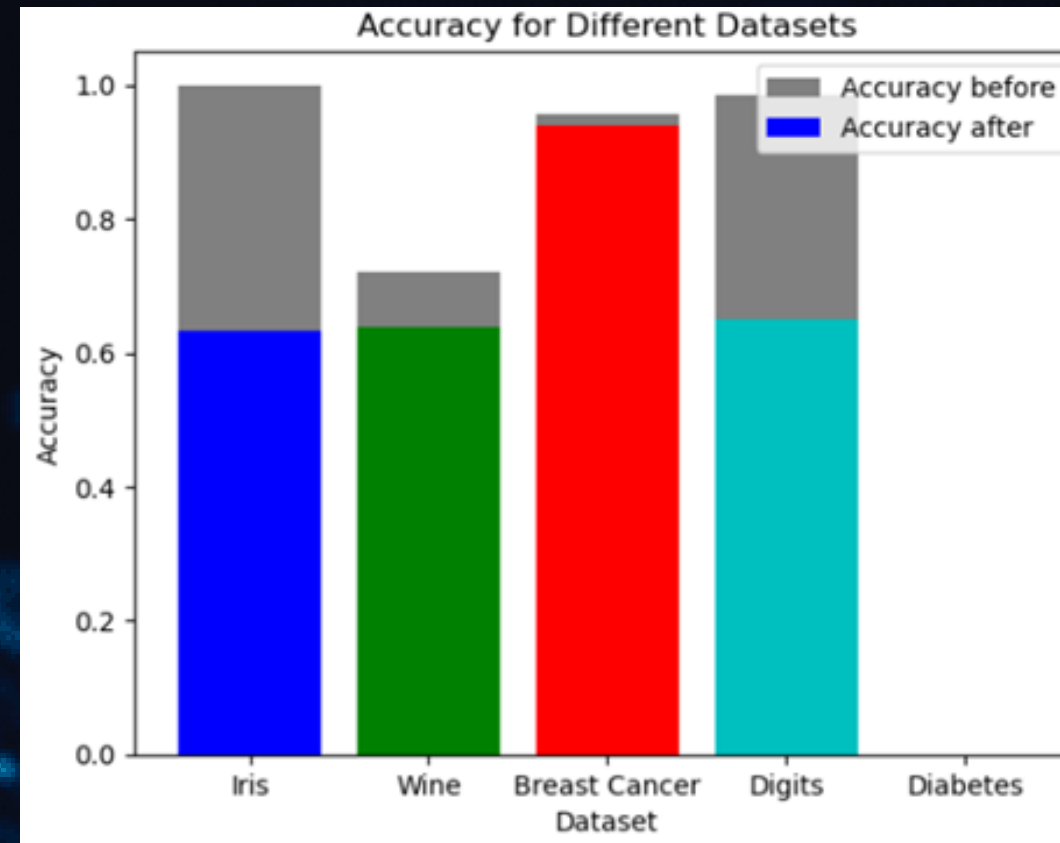
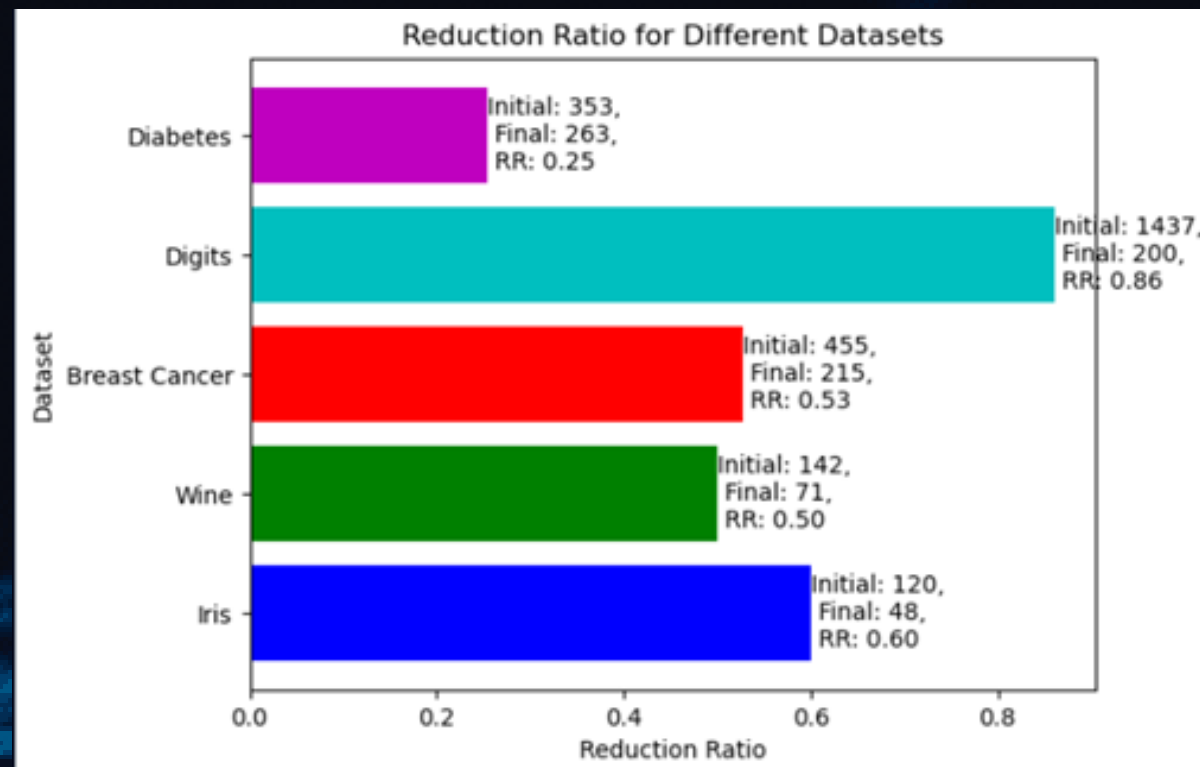
RMH

Condensed nearest neighbor(CNN)

- Prvi algoritam za selekciju instanci
- implementiran 1968. god.
- Dostupan u scikit.imbalanced-learn biblioteci
- Preporučen za manje setove podataka
- Inkrementalni algoritam
- Ne garantuje smanjenje pocetnog skupa podataka
- Može se koristiti za balansiranje podataka
- Kompleksnost max $O(n^3)$, n-broj instanci
- Pogodan za inkrementalno učenje

```
1 def condensed_k_nearest_neighbors(train):
2     samples = set()
3
4     # Choose a random starting observation
5     random_sample = train.pop(random.randint(0, len(train) - 1))
6     samples.add(random_sample)
7
8     additions = True
9     while additions:
10         additions = False
11         for _ in range(len(train)):
12             x = random.choice(train)
13
14             min_distance = float('inf')
15             closest_sample = None
16             for sample in samples:
17                 distance = x.calcDistance(sample)
18                 if distance < min_distance:
19                     min_distance = distance
20                     closest_sample = sample
21
22             if x.classifier != closest_sample.classifier:
23                 samples.add(x)
24                 train.remove(x)
25                 additions = True
26
27     print("Number of samples selected: " + str(len(samples)))
28     return samples
```

Rezultati primene CNN algoritma nad osnovnim skupovima podataka



Nedostaci CNN algoritma:

1. Izabrani skup podataka je u velikoj meri nasumičan
2. Vremenski zahtevan
3. Zahtevan u pogledu memorijskih resursa
4. Postoji mogućnost preterane adaptacije podskupa na trening skup
5. Ne radi dobro za podatke koji sadrže noise ili greške
6. Može smanjiti interpretabilnost podataka
7. Ne garantuje smanjenje broja instanci
8. Ne izbacuje duplikate iz trening skupa podataka

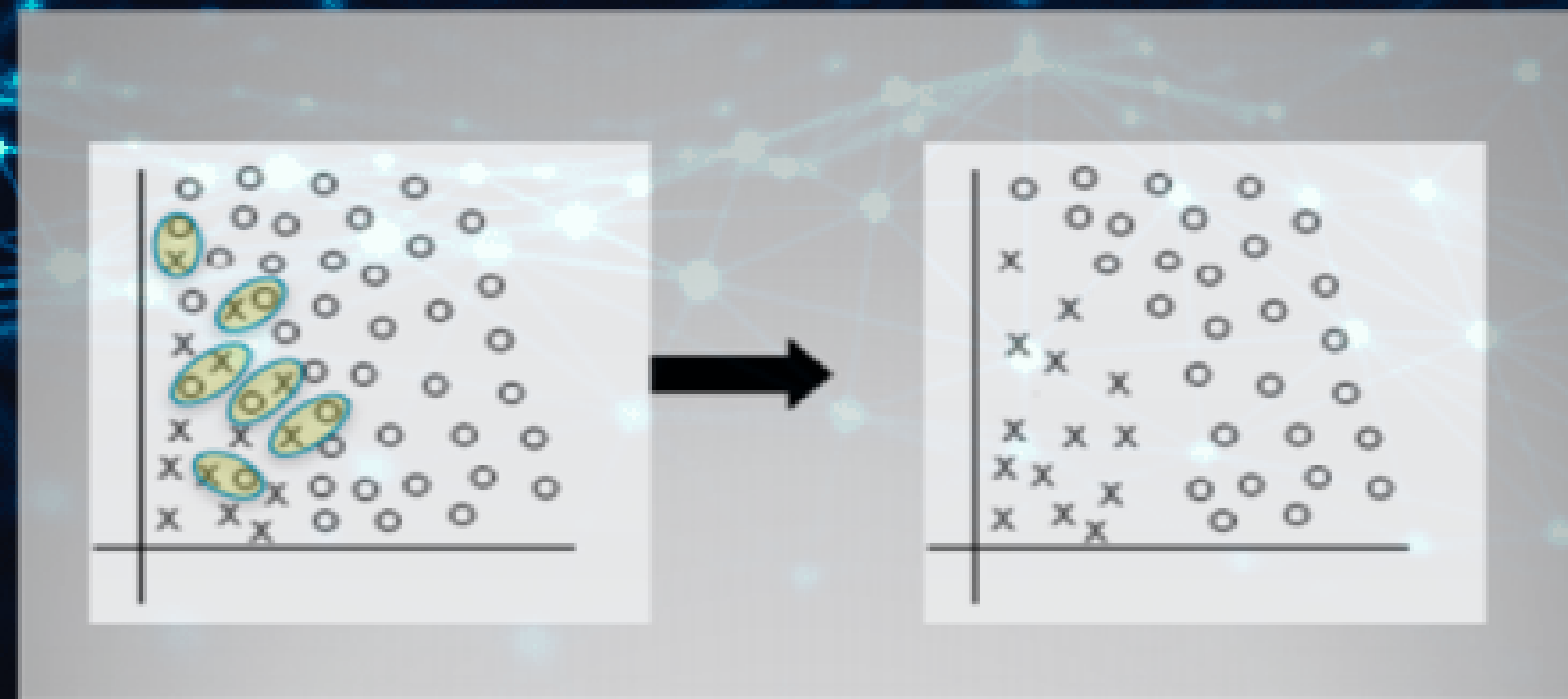
Prednosti CNN algoritma:

1. Smanjenje količine podataka
2. Sprečava overfitting

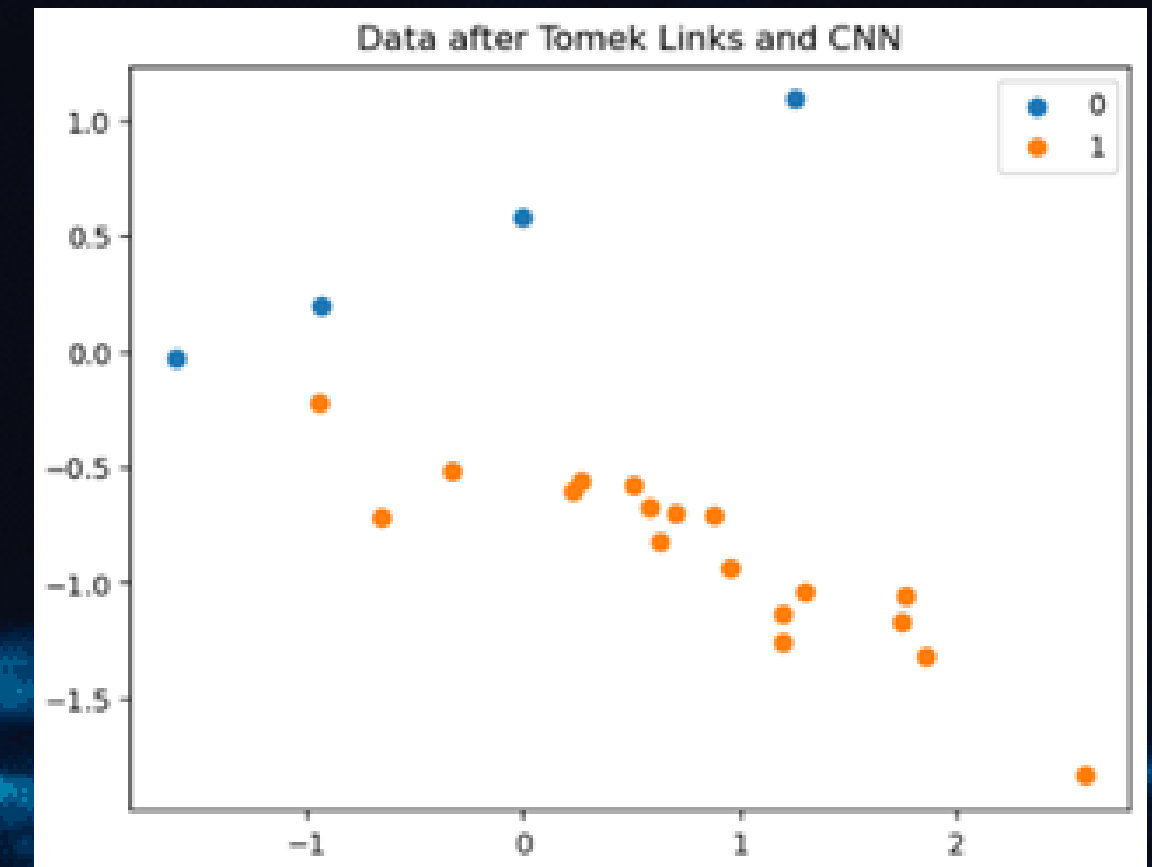
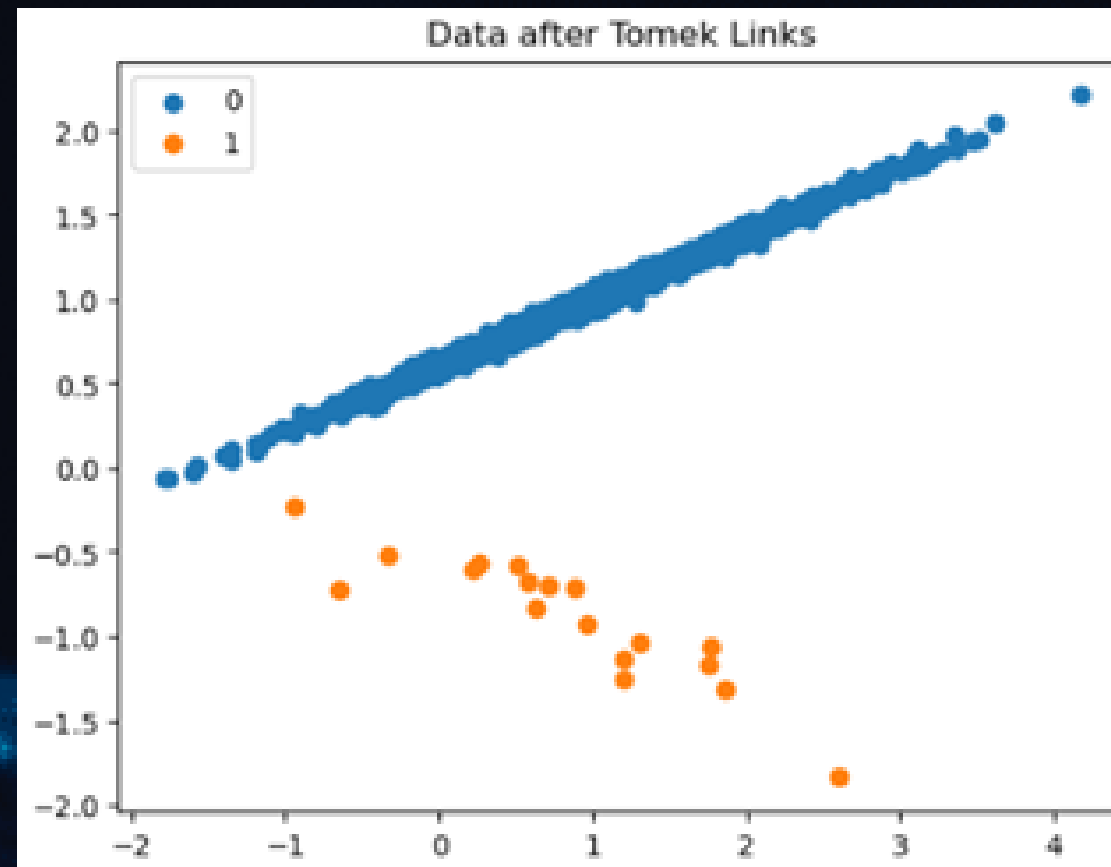
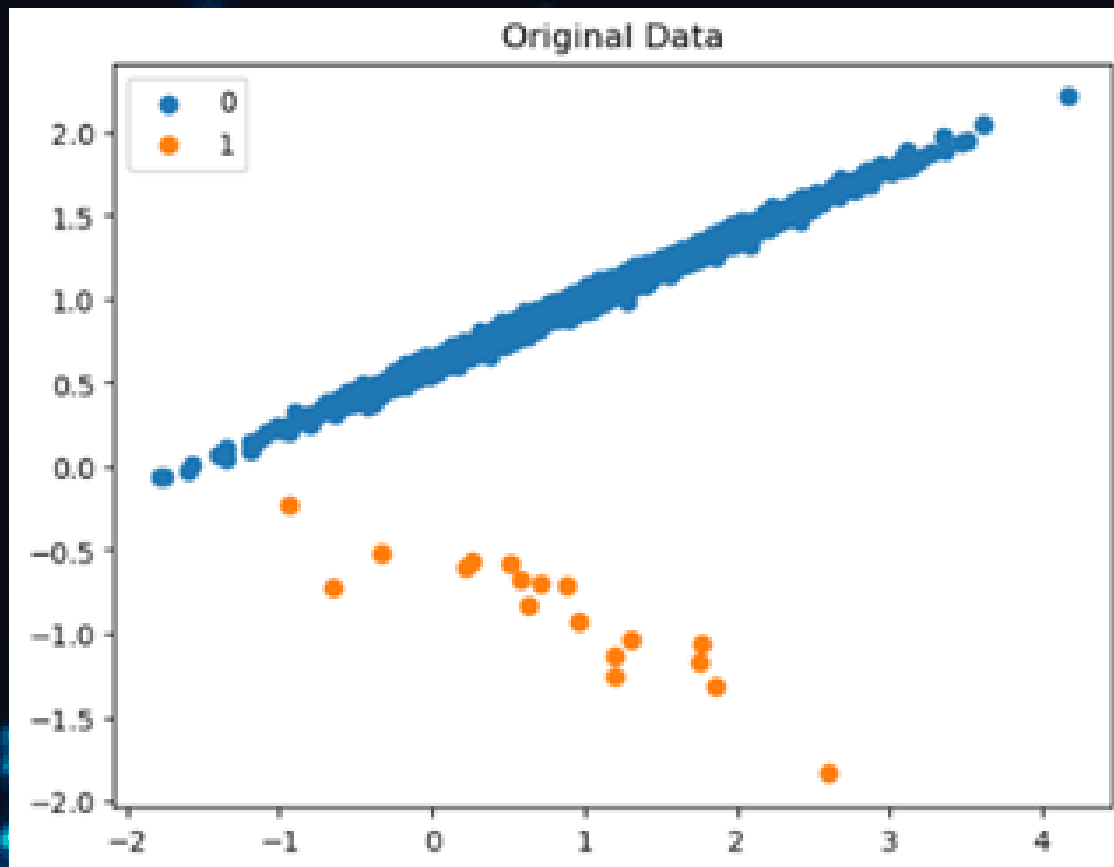
Unapređenje CNN - Tomek Condensed Nearest Neighbor(TCNN)

Dve osnovne modifikacije:

- Kada je instanca x suprotne klase od sebi najbliže instance y (slučaj kada dodajemo element x u podskup samples), umesto da se ta instanca doda u podskup, metod pronalazi instancu koja je suprotne klase od klase instance x , koja je najbliži sused instanci y (y i instanca koja se trazi pripadaju istoj klasi) i dodaje je u podskup samples.
- Pretpostavimo da u određenoj fazi generisanja redukovano skupa podataka E postoji instanca z koja se klasifikuje netačno jer nema tačaka iz neophodnog skupa A u E . U tom slučaju, pronađemo najbližeg suseda z u E , koji se zove u . Pošto se z klasifikuje netačno i jeste najbliži sused u , u mora pripadati suprotnoj klasi. Sada pronađemo najbližeg suseda u , koji se zove v , koji klasifikuje z ispravno. Instanca v pripada skupu A , koji je neophodan za tačnu klasifikaciju.



Primena TCNN i CNN



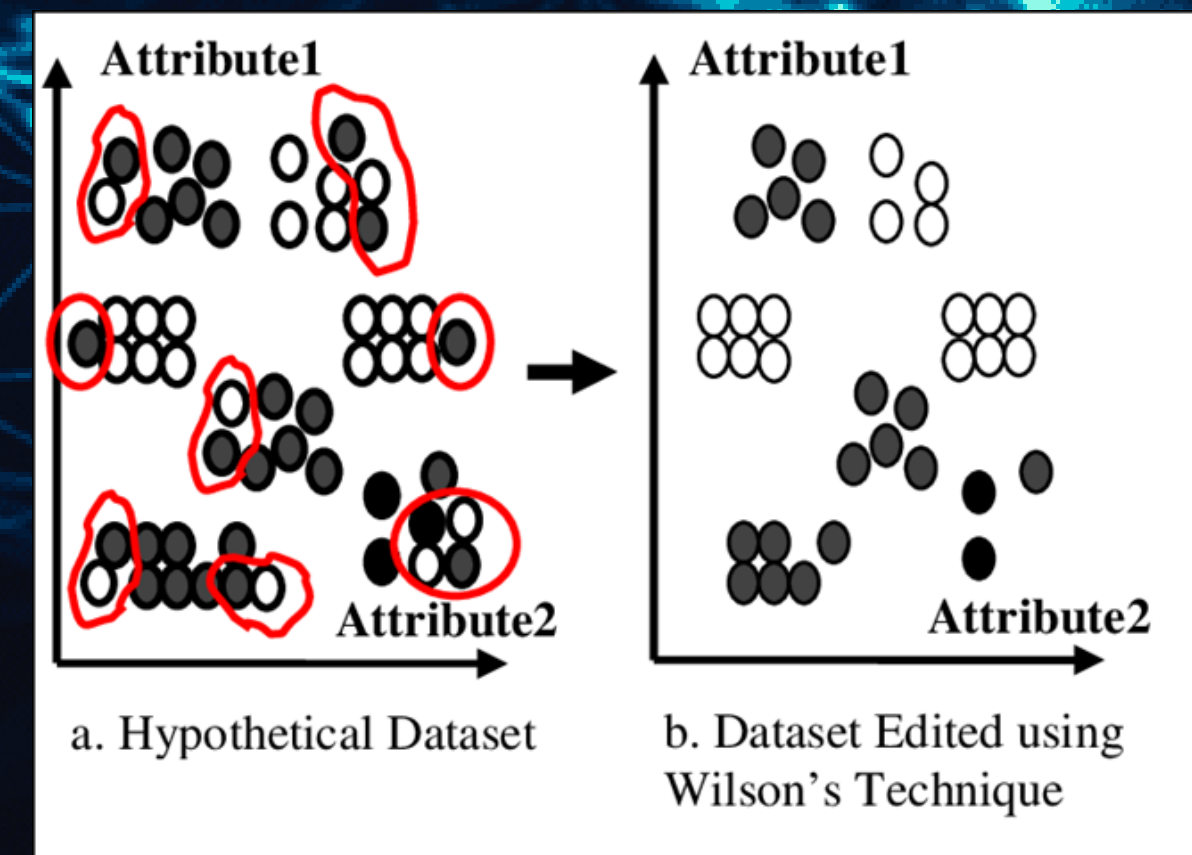
- Instance koje se nalaze u Tomek linkovima su ili granične instance ili noisy instance. To je zato što će samo granične instance i noisy instance imati najbliže susede koji pripadaju suprotnoj klasi.
-
- Izbor kombinovanja Tomek linkova i CNN-a je prirodan, jer se može reći da Tomek linkovi uklanjaju granične i noisy instance, dok CNN uklanja redundantne instance.

Edited Nearest Neighbour (ENN)

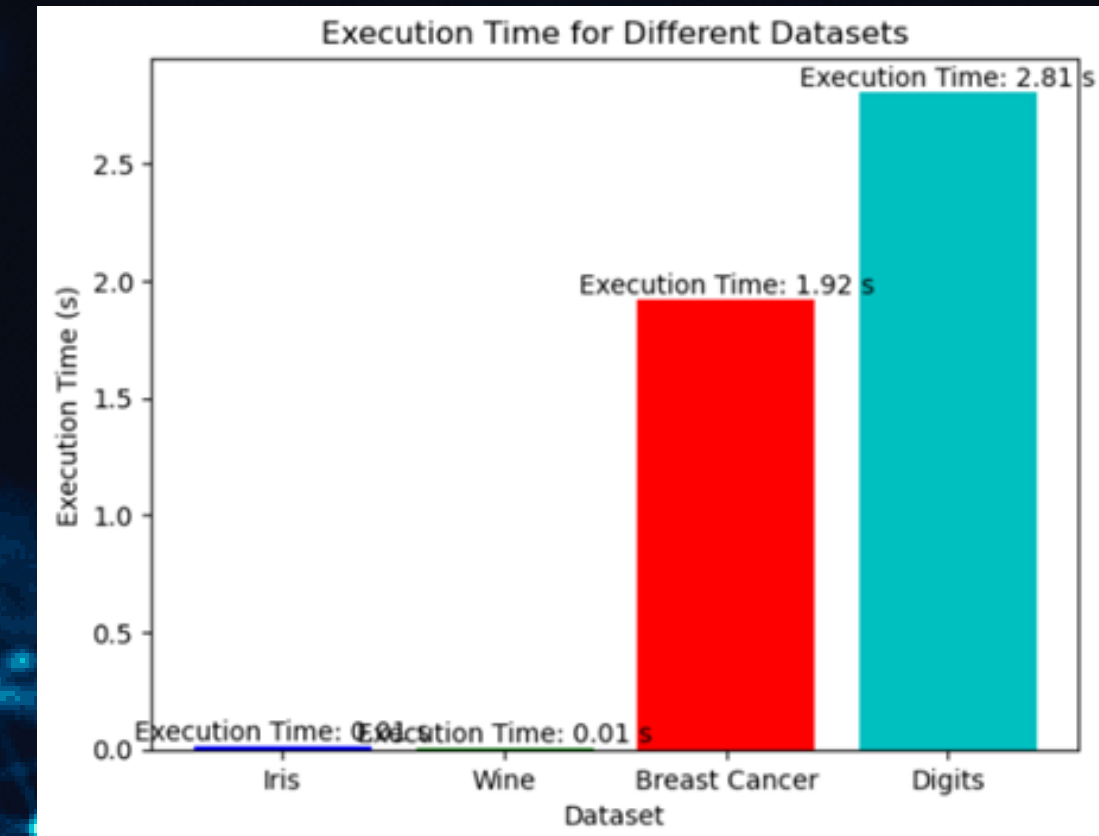
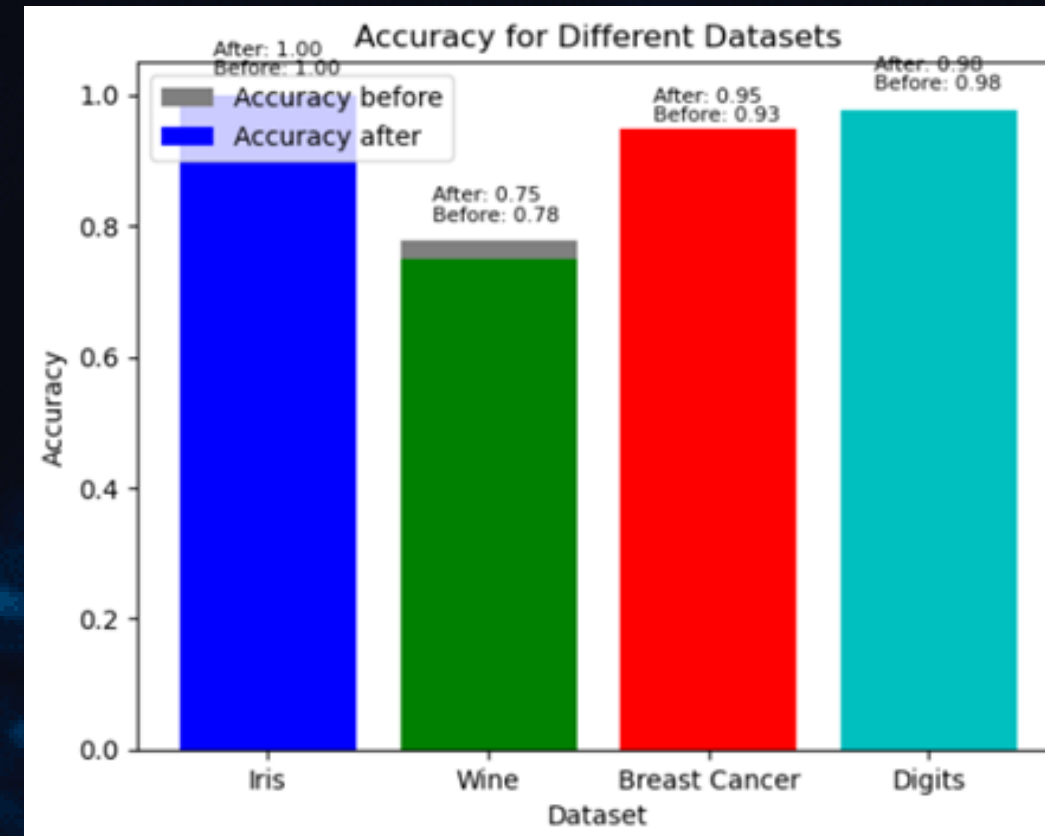
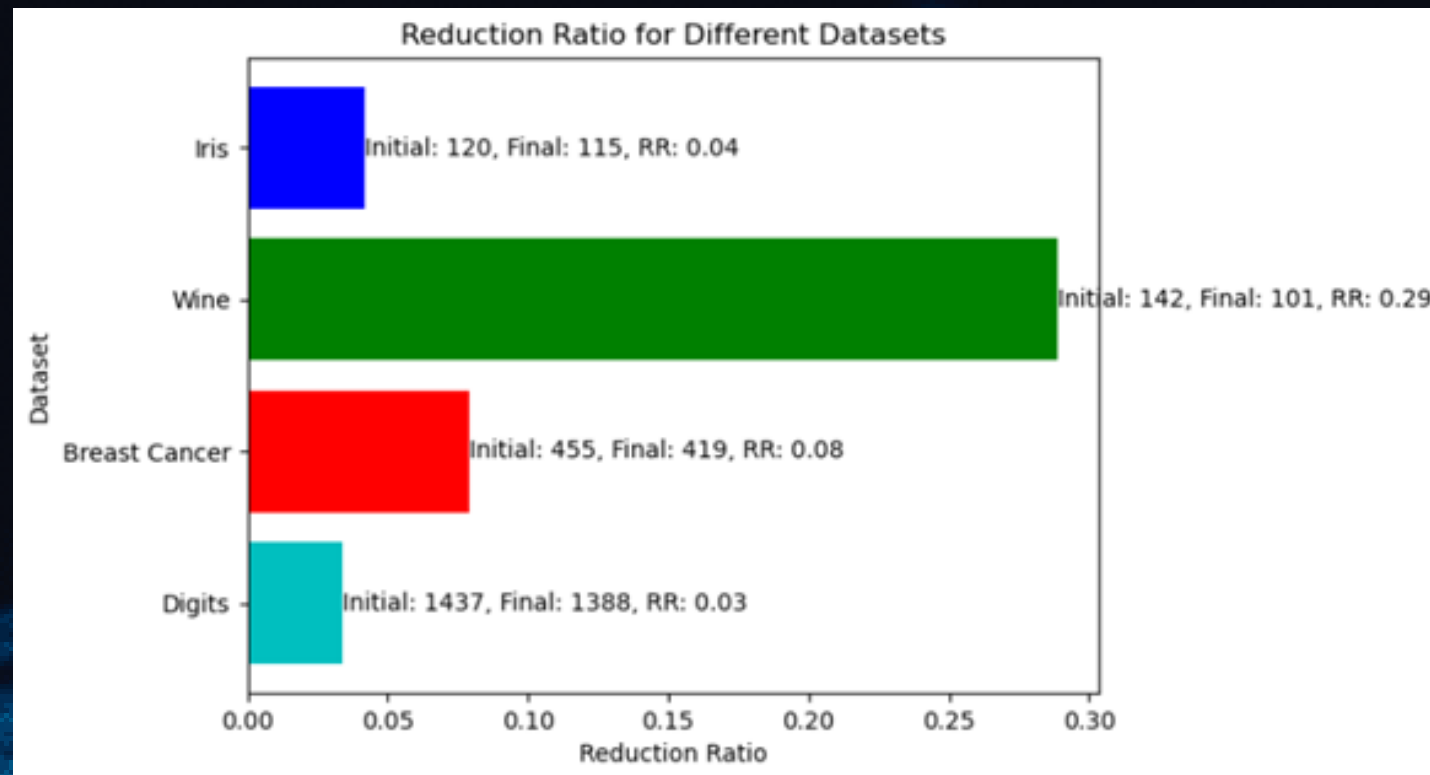
Koraci u ENN algoritmu su sledeći:

- Implementiran od strane David L. Wilsona
- Uklanja instance koje su verovatno pogresno klasifikovane
- Dostupan u scikit.imbalanced-learn biblioteci
- Preporučen za kompleksne setove podataka
- Dekrementalni algoritam
- Ne garantuje smanjenje pocetnog skupa podataka
- Smanjuje uticaj šuma i outliera
- Moze se koristiti za balansiranje podataka
- Kompleksnost $\max O(n \cdot (n+k))$, n -broj instanci, k - broj suseda

1. Dat je ulazni set trening podataka za N instanci, proverava se postavljena vrednost za K (broj najbližih suseda). Ako nije postavljeno K , uzima se difoltna vrednost 3.
2. Pronalazi se K najbližih suseda instance u setu podataka u odnosu na ostale instance.
3. Ako je instanca u klasi koja je različita od dominantne klase svoje okoline, instanca se izbacuje iz trening skupa podataka.
4. Ponavljati korake 2 i 3 dok se ne postigne željena posednutost klasa



Rezultati primene ENN algoritma nad osnovnim skupovima podataka



Prednosti:

1. Početni elementi nisu slučajno odabrani
2. Uklanja noisy podatke
3. Pozitivno utiče na tačnost klasifikacije

Nedostaci:

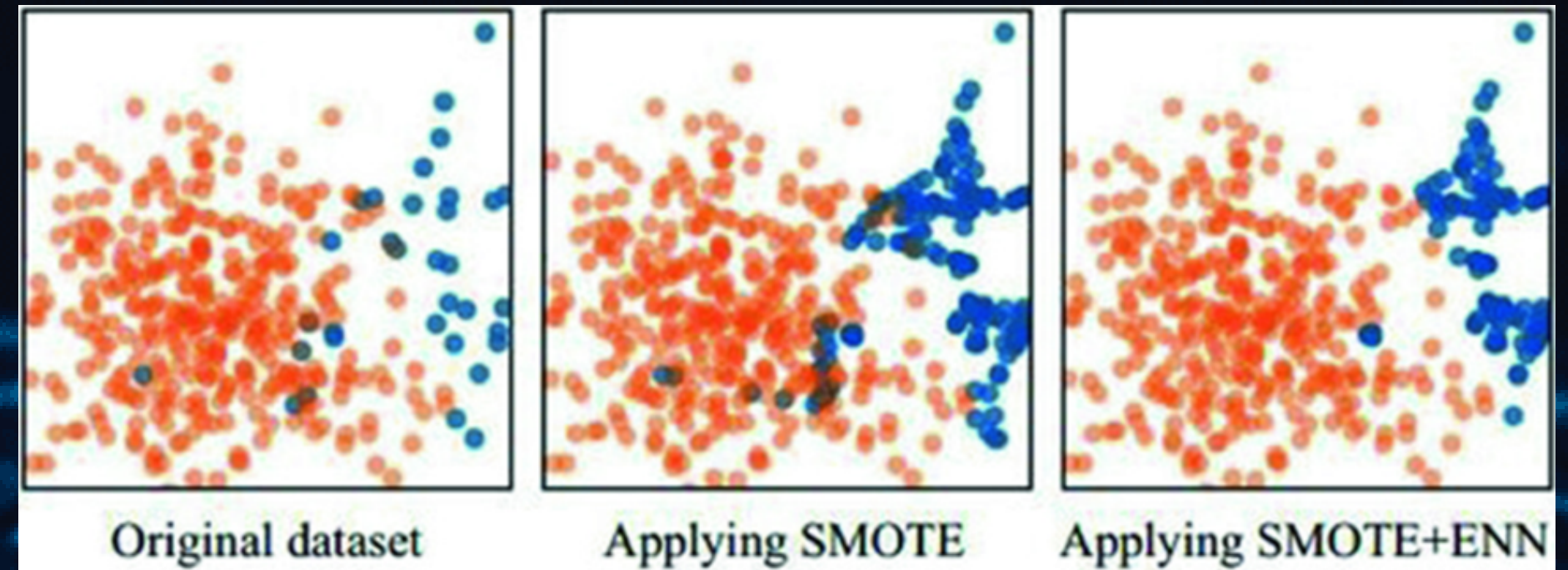
1. Neznatno smanjuje broj instanci
2. Sa povećanjem količine podataka, povećava se vreme potrebno za izvršenje algoritma

SMOTE + ENN - SMOTEENN

Ovaj metod, razvijen od strane Gustava Batiste 2004. god., kombinuje sposobnost SMOTE algoritma da generiše sintetičke primere za manje zastupljenu klasu i sposobnost ENN algoritma da obriše neke instance iz obe klase za koje je uočeno da imaju suprotnu klasu u odnosu na klasu njihovih K-najbližih suseda koji pripadaju većinskoj klasi

Početak SMOTE-a

1. Izabrati slučajne podatke iz manje zastupljene klase.
2. Izračunati udaljenost između slučajnih podataka i njihovih K-najbližih suseda.
3. Pomnožiti razliku sa slučajnim brojem između 0 i 1, a zatim dodati rezultat manje zastupljenoj klasi kao sintetički uzorak.
4. Ponavljati korake 2 i 3 sve dok se ne postigne željeni udeo manje zastupljene klase. (Kraj SMOTE-a)



Početak ENN-a

1. Odrediti K kao broj najbližih suseda. Ako nije određen, onda $K = 3$.
2. Pronaći K-najbližih suseda instance među ostalim instancama u skupu podataka, a zatim vrati većinsku klasu od K-najbližih suseda.
3. Ako klasa instance i većinska klasa njenih K-najbližih suseda nisu iste, onda se instance i njenih K-najbližih suseda brišu iz skupa podataka.
4. Ponavljati korake 2 i 3 sve dok se ne ispuni željeni udeo svake klase. (Kraj ENN-a)

Familija DROP (Decremental Reduction Optimization Procedure) algoritama - DROP1

Svaki algoritam iz ove familije ima sopstveno osnovno pravilo redukcije, kada je u pitanju DROP1 ono glasi:

- Ukloniti X_i ako bi bar isti broj njegovih pridruženih instanci u S bio klasifikovan ispravno bez X_i .

Kompleksnost ovog algoritma je $O(n \cdot (3k+2)) \sim O(n \cdot k)$.

```
function DROP1(TR):
    S = TR
    for each instance  $X_i$  in S:
        neighbors = find_k_nearest_neighbors( $X_i$ , S,  $k+1$ )
        for each neighbor in neighbors:
            add  $X_i$  to neighbor's list of associates
    for each instance  $X_i$  in S:
        with_ $X_i$  = count_correctly_classified( $X_i$ ,  $X_i$ .associates)
        without_ $X_i$  = count_correctly_classified( $X_i$ , S - { $X_i$ })
        if without_ $X_i$  >= with_ $X_i$ :
            S = S - { $X_i$ }
            for each associate in  $X_i$ .associates:
                remove  $X_i$  from associate's list of neighbors
                find new nearest neighbor for associate
                add associate to its new list of associates
            for each neighbor in  $X_i$ .neighbors:
                remove  $X_i$  from neighbor's list of associates
        end if
    return S
```


DROP2

Unapređena verzija DROP1 algoritma i izmenjeno osnovno pravilo:

- Ukloniti X_i ako bi bar isti broj njegovih pridruženih instanci u TR bio klasifikovan ispravno bez X_i .

DROP2 takođe menja redosled uklanjanja instanci. Inicijalno sortira instance u S prema udaljenosti do njihovog najbližeg protivnika. Provera za uklanjanje instanci započinje od instance najudaljenije od svog najbližeg protivnika.

```
function DROP2(TR):  
    S = TR  
    # Sort instances in S by distance to their nearest enemy  
    S = sorted(S, key=lambda x: x.distance_to_nearest_enemy, reverse=True)  
    for each instance  $X_i$  in S:  
        neighbors = find_k_nearest_neighbors( $X_i$ , S, k+1)  
        for each neighbor in neighbors:  
            add  $X_i$  to neighbor's list of associates  
    for each instance  $X_i$  in S:  
        with_ $X_i$  = count_correctly_classified( $X_i$ ,  $X_i$ .associates)  
        without_ $X_i$  = count_correctly_classified( $X_i$ , S - { $X_i$ })  
        if without_ $X_i$  >= with_ $X_i$ :  
            S = S - { $X_i$ }  
        else:  
            for each associate in  $X_i$ .associates:  
                if associate not in S:  
                    remove  $X_i$  from associate's list of neighbors  
                    find new nearest neighbor for associate  
                    add associate to its new list of associates  
            for each neighbor in  $X_i$ .neighbors:  
                if neighbor not in S:  
                    remove  $X_i$  from neighbor's list of associates  
    return S
```

DROP3 - CPruner

- Ovaj algoritam je kombinacija DROP2 i ENN algoritma, takođe ima dosta izmena u odnosu na prethodne algoritme u ovoj familiji.

Pravilo(rule) 1. “Odsecanje” instance (eng. instance pruning rule).

Instanca X_i u TR se može odseci ako zadovoljava jedan od sledeća dva uslova:

- -Ona je noisy instance
- -Ona je suvisna (superfluous) instanca, ali nije kritična (critical)

Novi pojmovi:

- k-reachability
- k-coverage
- superfluous instanca
- critical instanca
- noisy instanca

Pravilo(rule) 2. za određivanje redosleda uklanjanja instanci.

Neka $H\text{-}k\text{NN}(X_i)$ bude broj instanci njegove klase u $k\text{NN}(X_i)$, i $D\text{-NE}(X_i)$ bude udaljenost X_i do njenog najbližeg neprijatelja.

Za dve odsecive instance X_i i X_j u TR:

- -Ako je $H\text{-}k\text{NN}(X_i) > H\text{-}k\text{NN}(X_j)$, X_i treba ukloniti pre X_j
- -Ako je $H\text{-}k\text{NN}(X_i) = H\text{-}k\text{NN}(X_j)$ i $D\text{-NE}(X_i) > D\text{-NE}(X_j)$, X_j treba ukloniti pre X_i
- -Ako je $H\text{-}k\text{NN}(X_i) = H\text{-}k\text{NN}(X_j)$ i $D\text{-NE}(X_i) = D\text{-NE}(X_j)$, redosled uklanjanja se odlučuje nasumično.



```
S = T*R
```

```
# Step 1: Compute k-reachability and k-coverage for all instances
```

```
for Xi in S:
```

```
    k_reach = compute_k_reachability(Xi)
```

```
    k_cov = compute_k_coverage(Xi)
```

```
# Step 2: Remove noisy instances and update k-reachability and k-coverage
```

```
for Xi in S:
```

```
    if Xi.is_noisy():
```

```
        S.remove(Xi)
```

```
        for Xj in k_cov[Xi]:
```

```
            k_reach[Xj].remove(Xi)
```

```
            update_k_reachability(Xj)
```

```
        for Xj in k_reach[Xi]:
```

```
            k_cov[Xj].remove(Xi)
```

```
# Step 3: Sort the instances in S according to rule 2
```

```
S = sorted(S, key=lambda Xi: rule2(Xi))
```

```
# Step 4: Remove instances that satisfy rule 1 and update k-reachability
```

```
for Xi in S:
```

```
    if rule1(Xi):
```

```
        S.remove(Xi)
```

```
        for Xj in k_cov[Xi]:
```

```
            k_reach[Xj].remove(Xi)
```

```
            update_k_reachability(Xj)
```

```
# Step 5: Return the final set of instances S
```

```
return S
```


	CNN	ENN	DROP	Tomek-Links	SMOTEE NN
<u>Smer pretrage</u>	Incremental	Decremental	Decremental	Decremental	Decremental
<u>Tip selekcije</u>	Condensed	Edition	Hybrid	Condensed	Hybrid
<u>Evaluacija pretrage</u>	/	/	Filter	/	/
<u>Redukcija dimenzionalnosti</u>	<u>Velika</u>	Mala	<u>Srednja</u>	Mala	<u>Srednja</u>
<u>Uklanjanje <i>noisy</i> instance</u>	Ne	Da	Da	Ne	Da
<u>Povećava/smanjuje/ne menja accuracy</u>	Ne menja/smanjuje	<u>Povećava/smanjuje/ne menja</u>	<u>Povećava/Ne menja</u>	<u>Povećava/Ne menja</u>	<u>Povećava/Ne menja</u>
<u>Vremenski zahtevan</u>	Da	Ne	Ne	Ne	Ne
<u>Kompleksnost</u>	$O(n^3)$	$O(n*(k+n))$	$O(n*k)$	$O(n)$	$O(n*k)$
<u>Sprečava overfitting</u>	Da	Ne	Ne	Ne	Ne
<u>Garantuje redukciju</u>	Ne	Ne	Ne	Ne	Ne
<u>Izbacuje duplikate</u>	Ne	Ne	Ne	Ne	Ne
<u>Nasumičan rad</u>	Da	Ne	Ne	Ne	NE

Zaključak

- Instance selection algoritmi predstavljaju moćan "alat" za redukciju dimenzionalnostii
- Glavni nedostatak jeste to što za većinu algoritama nema dostupnih implementacija
- Problem predstavlja i povezanost sa bigdata oblasću, jer na drugi način rešava dati problem, pa se ova oblast manje istražuje

Rang lista efikasnosti algoritama:

1. SMOTEENN
2. NCR
3. ENN
4. TomekLinks
5. CNN



Hvala

na pažnji!