3. Laboratorijska vaja razpoznavanje vzorcev

Mentor: as. dr. Klemen Grm, mag. inž. el.

Avtor: Blaž Ardaljon Mataln Smehov

Datum: 27.12.2021

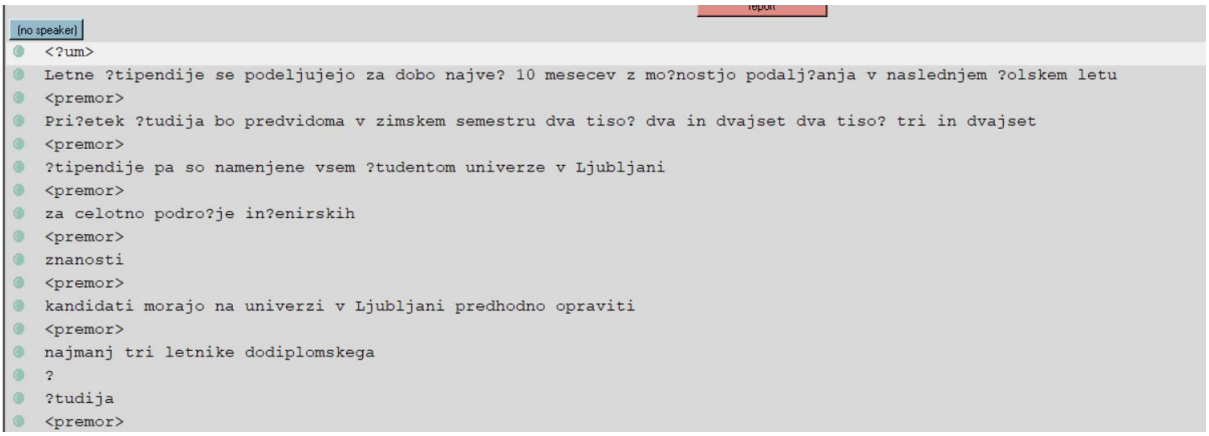
Vaja 3: Razpoznavanje glasov slovenskega jezika

Pri zadnji laboratorijski vaji smo morali zapisati program za razpoznavanje glasov slovenskega jezika. Za izvajanje vaje smo že dobili predlogo kode, v kateri smo morali dopolniti določene funkcije.

Za izvajanje vaje smo morali najprej pripraviti zvočni posnetek po vnaprej zapisanih navodilih.

Prvo sem izbral ustrezno slovnično pravilno besedilo, za to sem uporabil članke, ki so objavljeni na spletni strani fakultete. Zvočni posnetek sem posnel v programu Audacity in ga shranil kot 16-bitno wav datoteko. V navodilih je sicer pisalo, da mora biti posnetek 32-bitni, vendar v taki obliki v Transcriberju nisem videl valovnega zapisa posnetka.

V drugem koraku sem v Transcriberju moral po odsekih, ki sem jih določil sam zapisati kaj sem v tem odseku govoril, ali dodati odseku komentar v primeru da nisem govoril. Primer takega komentarja je na primer <premor>



Ko sem dokončal prepisovanje besedila sem se moral povezati na šolski računalnik, da sem lahko ti dve datoteki spremenil v primeren format, ki ga lahko potem obdelujemo v Pythonu. Končni cilj je arff datoteka.

Po končanem snemanju in pretvorbi sem se lotil programiranja.

Prvi del programa, ki smo ga morali dopolniti je funkcija:

```
def pripravi_zbirko(filename)
```

V tej funkciji smo morali v posamezne sezname shraniti značilke vzorcev, foneme in oznake vzorcev.

Shranjevanje fonemov: te sem shranil tako, da sem na internetu našel spletno mesto na katerem si lahko ogledaš arff datoteko. Enostavno sem poiskal del v katerem so zapisani fonemi in ga prekopiral v program:

```
fonemi =
"@,E,O,S,W,a,b,d,e,f,g,i,j,k,l,m,n,o,p,r,s,sil,t,tlesk,ts,u,v,vdih,w,x,z".spli
t(",")
```

pri tem sem uporabil funkcijo .split(','). Ta funkcija razdeli seznam na elemente, kaj je posamičen element določi glede na to kaj dodamo v oklepaju, v tem primeru je v oklepaju zapisana vejica, kar pomeni, da vsakič ko program prebere vejico je naslednja prebrana stvar nov element. V tem primeru to pomeni da je vsak znak svoj element.

V drugem delu sem zapisal kodo, ki shrani značilke vzorcev:

```
    zacetek = vrstice.index("@DATA")
    konec = vrstice.index(vrstice[-1])
    for i in range(zacetek+1, konec+1):
        seznam = vrstice[i].split(",")
        y_seznam.append(seznam[-1])
        X_seznam.append(seznam[0:-1])
```

Če bi si ogledali arff datoteko bi videli, da se značilke začnejo za vrstico @DATA, v predlogi kode je že podan seznam vrstice. Vsak element tega seznama je ena vrstica iz arff datoteke. Torej najprej sem poiskal mesto indeksa kjer se začnejo značilke (za @DATA), to sem storil s funkcijo .index, ki ti vrne številko mesta, oziroma indeks stringa ali katere druge spremenljivke, ki jo podaš. Konec značilk je seveda konec seznama, torej sem poiskal tudi indeks konca seznama.

V arff datoteki vidimo, da je vsaka vrstica sestavljena iz več elementov, ki so ločeni z vejicami. Seznam značilk smo določili tako, da smo vse te elemente ločili med seboj s funkcijo split, ki loči vse elemente med katerimi so vejice. Konec vsake vrstice je podana oznaka vzorcev. Značilke sem shranil posebej od oznak. Kot je vidno sem vse elemente razen zadnjega torej oznake shranil s funkcijo append v seznam x, vse oznake pa sem z appendom shranil v seznam y.

Funkcija:

```
.append
```

Ti shrani podan element v oklepaju na konec seznama, ki ga navedeš prek funkcijo.

Naslednja funkcija, ki smo jo morali dopolniti je:

```
def konfuzijska_matrika(y_test, y_hat, fonemi):
    """Prikaže konfuzijsko matriko razpoznavalnika na podlagi
    pravilnih oznak (y_test), izračunanih oznak (y_hat) ter seznama
fonemov."""
```

Sestaviti smo morali konfuzijsko matriko. Indeks konfuzijske matrike je vrednost y_test na x osi in vrednost y_hat na y osi. Vsakič, ko se ponovi kombinacija vrednosti istoležnih elementov teh dveh seznamov se v matriko vpiše vrednost 1:

```
for i in range(len(y_test)):
    matrika[y_test[i],y_hat[i]] += 1
```

vrednosti sem nato povprečil, tako da sem vsak element matrike delil z vsoto vseh elementov in s funkcijo np.diagonal iz knjižnice numpy dobil diagonalo povprečene matrike:

```
vsota_matrike = np.sum(matrika)
nova_matrika = matrika/vsota_matrike

diagonala_fonemov = np.diagonal(nova_matrika)
```

Iz diagonale lahko dobimo najboljše prepoznan fonem in najslabše prepoznan fonem, s funkcijama iz knjižnice numpy np.argmax in np.argmin. Ti dve funkciji nam vrneta indeks, oziroma mesto v seznamu, kjer se nahajata največja in najmanjša vrednost.

```
najbolj_prepoznan_f = np.argmax(diagonala_fonemov)
    najmanj_prepoznan_f = np.argmin(diagonala_fonemov)
```

V naslednjem koraku sem potem še izračunal najpogostejšo napako za posamičen fonem. Torej z for zanko sem šel skozi vsak fonem v seznamu fonemov. Funkcija iz knjižice np.argsort ti razvrsti elemente po velikosti, če vzamemo zadnji element dobimo najpogostejšo napako:

```
for i, element in enumerate(nova_matrika):
    indeks = np.argsort(element)[::-1]
    print("za fonem: ", fonemi[i])
    if i != indeks[0]:
        print("najpogostejša napaka: ", fonemi[indeks[0]])
    else:
        print("najpogostejša napaka: ", fonemi[indeks[1]])
```

Za konec smo morali v mainu še klicati zapisane funkcije in preveriti delovanje razvrščevalnikov iz knjižnice sklearn.

Klic funkcije, ki nam prebere arff datoteko in nam sestavi sezname fonemov, značilk in oznak:

```
X, y, fonemi = pripravi_zbirko("govorec1.arff")
```

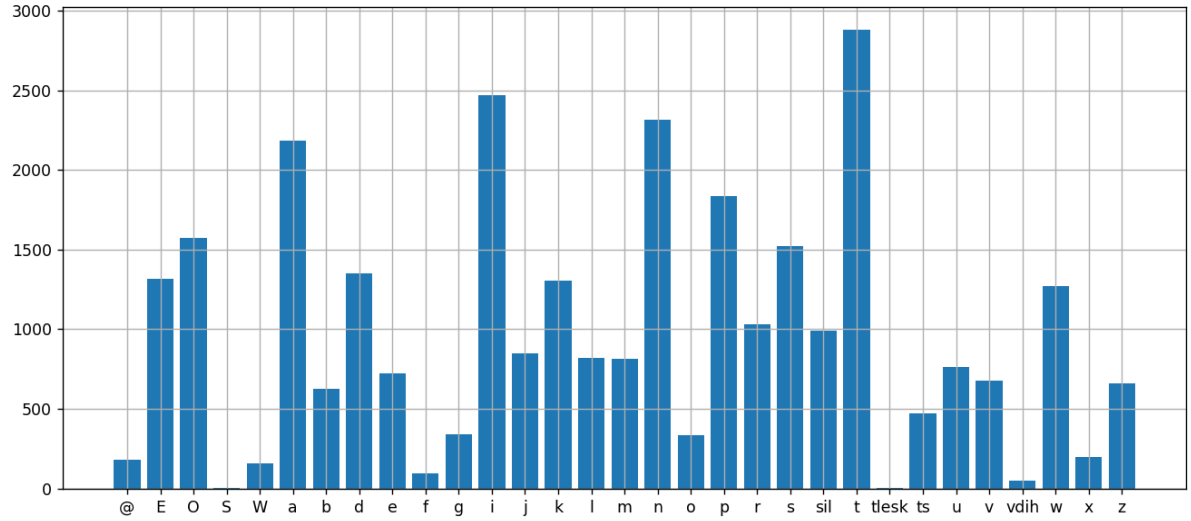
Pri metodi najbližjih sosedov moramo predhodno podati k-je za katere želimo računati, ti morajo biti liha števila. Ker računanje dane programske kode na mojem računalniku traja izjemno dolgo sem se omejil na manjše število variacij računanja uspešnosti metod. Pri metodi najbližjih sosedov sem izbral 5 različnih k-jev. Kot metodo računanja sem izbral euclidean, če si na spletu pogledamo funkcijo KNeighborsClassifier vidimo, da je možnih več metricsov. Uspešnost smo v tej for zanki in v vseh naslednjih z drugo metodo računali s klicem funkcije, ki je že bila podana v predlogi kode.

Naslednja metoda je svm. Nastaviti je možno več parametrov, v navodilih za vajo so omenjeni C, jedro, stopnja in gamma. Če pogledamo na spletu informacije o funkciji lahko vidimo katere vrednosti lahko vstavimo za te parametre. Jaz sem poskusil različne C-je (regulacijski parameter?(mogoče)), gamma je v obeh for zanskah scale. Stopnja je enkrat 2, enkrat 3. Jedro je enkrat rbf, enkrat poly. Kot sem že omenil uspešnost je ponovno zračunana s klicem prej zapisane funkcije.

```
seznam_k = [1,3,5,7,9]

    for k in seznam_k:
        najblizji_sosed = KNeighborsClassifier(n_neighbors=k,
metric='euclidean')
        uspesnosti = navzkrizno_preverjanje(X, y, 5, najblizji_sosed, fonemi)
        povprecna_uspesnost = np.mean(uspesnosti)
        print("uspesnost KNN: ", uspesnosti)
        print("povprecna uspesnost: ", povprecna_uspesnost)

    for c in [1, 10, 100, 1000]:
        svm1 = SVC(C = c,kernel = 'rbf', degree = 3, gamma = 'scale')
        uspesnosti1 = navzkrizno_preverjanje(X, y, 5, svm1, fonemi)
        povprecna_uspesnost1 = np.mean(uspesnosti1)
        print("uspesnost SVM s C-ji: ", uspesnosti1)
        print("povprecna uspesnost s C-ji, rbf, degree 3: ",
povprecna_uspesnost1)

    for c in [1, 10, 100, 1000]:
        svm2 = SVC(C = c,kernel = 'poly', degree = 2, gamma = 'scale')
        uspesnosti2 = navzkrizno_preverjanje(X, y, 5, svm2, fonemi)
        povprecna_uspesnost2 = np.mean(uspesnosti2)
        print("uspesnost SVM s C-ji: ", uspesnosti2)
        print("povprecna uspesnost s C-ji, poly, degree 2: ",
povprecna_uspesnost2)
```

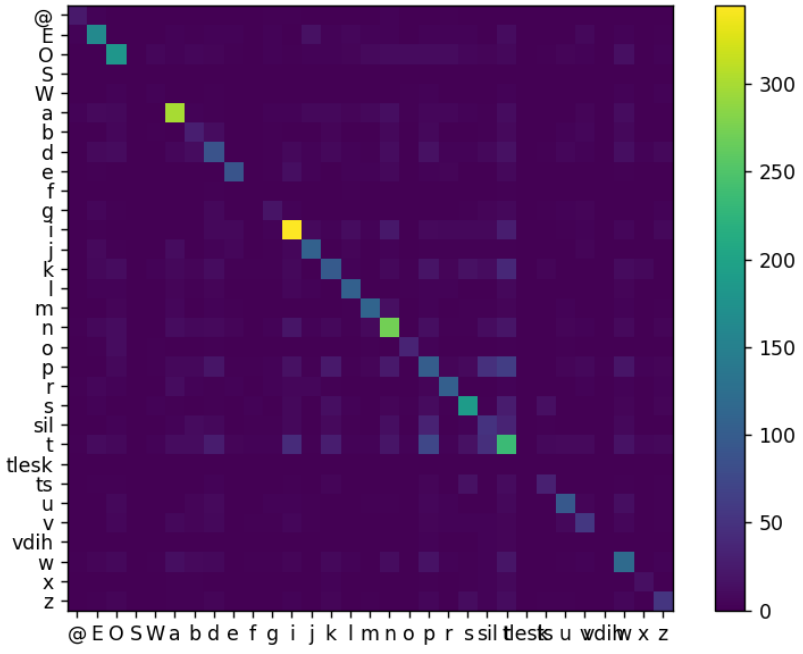
Rezultati:

Histogram fonemov:



Najbližji sosed:

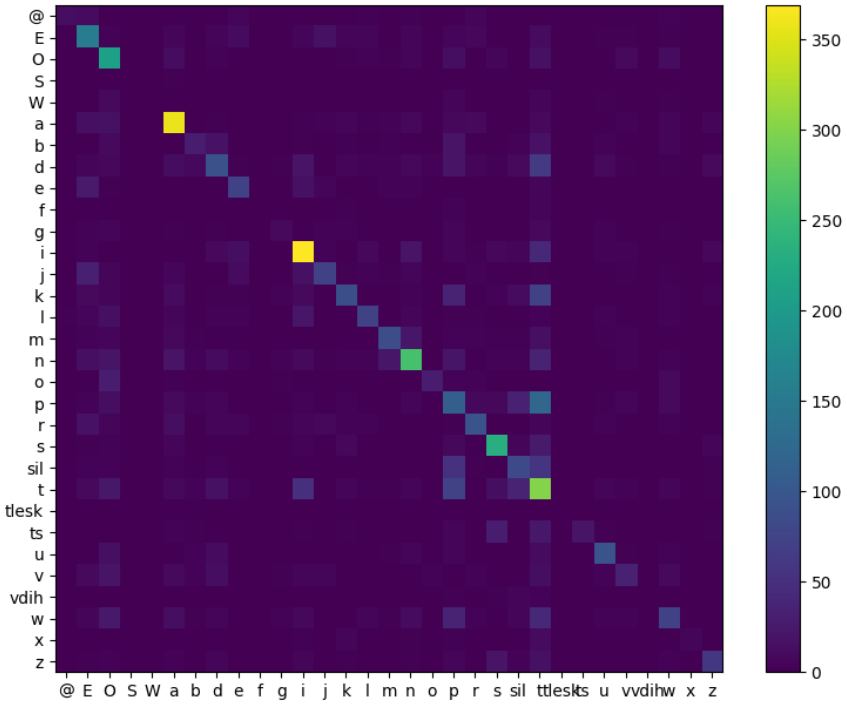
```
najbolje prepoznan: i
najmanj prepoznan: S
za fonem:  @
najpogostejša napaka:  n
za fonem:  E
najpogostejša napaka:  j
za fonem:  O
najpogostejša napaka:  w
za fonem:  S
najpogostejša napaka:  z
za fonem:  W
najpogostejša napaka:  O
za fonem:  a
najpogostejša napaka:  n
za fonem:  b
najpogostejša napaka:  d
za fonem:  d
najpogostejša napaka:  p
za fonem:  e
najpogostejša napaka:  i
za fonem:  f
najpogostejša napaka:  t
za fonem:  g
najpogostejša napaka:  d
za fonem:  i
najpogostejša napaka:  t
za fonem:  j
najpogostejša napaka:  a
za fonem:  k
najpogostejša napaka:  t
za fonem:  l
najpogostejša napaka:  t
za fonem:  m
najpogostejša napaka:  n
za fonem:  n
najpogostejša napaka:  t
za fonem:  o
najpogostejša napaka:  O
za fonem:  p
najpogostejša napaka:  t
za fonem:  r
najpogostejša napaka:  a
za fonem:  s
najpogostejša napaka:  t
za fonem:  sil
najpogostejša napaka:  t
za fonem:  t
najpogostejša napaka:  p
za fonem:  tlesk
najpogostejša napaka:  z
za fonem:  ts
najpogostejša napaka:  s
za fonem:  u
najpogostejša napaka:  w
za fonem:  v
```




```
uspesnost KNN:  [0.49589132986751633, 0.5037732684890156, 0.49337581754150595, 0.5084688914975684, 0.4905249035720275]
povprecna uspesnost:  0.49840684219352677
```

SVM: rbf, degree = 3:

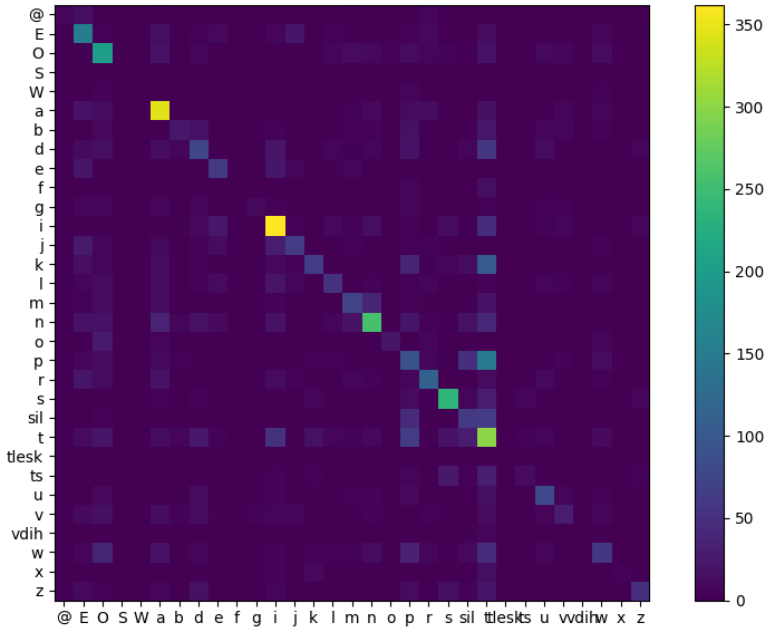
```
najpogostejša napaka:  a
za fonem:  W
najpogostejša napaka:  O
za fonem:  a
najpogostejša napaka:  O
za fonem:  b
najpogostejša napaka:  p
za fonem:  d
najpogostejša napaka:  t
za fonem:  e
najpogostejša napaka:  E
za fonem:  f
najpogostejša napaka:  t
za fonem:  g
najpogostejša napaka:  t
za fonem:  i
najpogostejša napaka:  t
za fonem:  j
najpogostejša napaka:  E
za fonem:  k
najpogostejša napaka:  t
za fonem:  l
najpogostejša napaka:  i
za fonem:  m
najpogostejša napaka:  n
za fonem:  n
najpogostejša napaka:  t
za fonem:  o
najpogostejša napaka:  O
za fonem:  p
najpogostejša napaka:  t
za fonem:  r
najpogostejša napaka:  E
za fonem:  s
najpogostejša napaka:  t
za fonem:  sil
najpogostejša napaka:  t
za fonem:  t
najpogostejša napaka:  p
za fonem:  tlesk
najpogostejša napaka:  sil
za fonem:  ts
najpogostejša napaka:  s
za fonem:  u
najpogostejša napaka:  O
za fonem:  v
najpogostejša napaka:  O
za fonem:  vdih
najpogostejša napaka:  sil
za fonem:  w
najpogostejša napaka:  t
za fonem:  x
najpogostejša napaka:  t
za fonem:  z
najpogostejša napaka:  s
```




```
uspesnost SVM s C-ji:  [0.46050645648163674, 0.46168036223377495, 0.4626865671641791, 0.4626865671641791, 0.4804628542679859]
povprecna uspesnost s C-ji, rbf, degree 3:  0.4656045614623512
```

SVM: poly, degree = 2:

```
za fonem:  E
najpogostejša napaka:  j
za fonem:  O
najpogostejša napaka:  a
za fonem:  S
najpogostejša napaka:  z
za fonem:  W
najpogostejša napaka:  p
za fonem:  a
najpogostejša napaka:  E
za fonem:  b
najpogostejša napaka:  t
za fonem:  d
najpogostejša napaka:  t
za fonem:  e
najpogostejša napaka:  i
za fonem:  f
najpogostejša napaka:  t
za fonem:  g
najpogostejša napaka:  O
za fonem:  i
najpogostejša napaka:  t
za fonem:  j
najpogostejša napaka:  i
za fonem:  k
najpogostejša napaka:  t
za fonem:  l
najpogostejša napaka:  i
za fonem:  m
najpogostejša napaka:  n
za fonem:  n
najpogostejša napaka:  t
za fonem:  o
najpogostejša napaka:  O
za fonem:  p
najpogostejša napaka:  t
za fonem:  r
najpogostejša napaka:  E
za fonem:  s
najpogostejša napaka:  t
za fonem:  sil
najpogostejša napaka:  t
za fonem:  t
najpogostejša napaka:  p
za fonem:  tlesk
najpogostejša napaka:  z
za fonem:  ts
najpogostejša napaka:  t
za fonem:  u
najpogostejša napaka:  t
za fonem:  v
najpogostejša napaka:  O
za fonem:  vdih
najpogostejša napaka:  t
za fonem:  w
```




```
uspesnost SVM s C-ji:  [0.44474257923863825, 0.4645312762032534, 0.44457487841690424, 0.46050645648163674, 0.45010900553412714]
povprecna uspesnost s C-ji, poly, degree 2:  0.45289283917491197
```