

Kolokvijum 2 – Priprema

Uvod [nadgledano učenje]

Poenta nadgledanog učenja je da imamo **LABELU** !

Voditi računa da ako dobijemo da rešavamo problem nadgledanog učenja, nikako ne krenemo da rešavamo K-means jer je to problem nenadgledanog učenja, ako to uradimo, **automatski smo pali kolokvijum** !

Moramo razumeti šta su labelirani podaci a šta nisu.

Znači, ako predviđamo da li neko ima rak kože, moramo imati **LABELU** o tome da li je neko imao ili nije imao rak kože. Takodje, ako predviđamo da li je neko muško ili žensko, mi moramo imati atribut u data setu da li je muško ili žensko kako bi imali na **ČEMU DA UČIMO** !

Još jedan primer, imamo podatke o ljudima iz bolnice, nivo trombocita, krvnih zrnaca I čega sve ne, I pokušavamo da utvrdimo da li pacijent ima Covid 19 ili ne, mi ne možemo obučiti model ako nemamo **LABELU** koja nam kaže da li je određeni entitet *imao ili nije imao* Covid19 .

Voditi računa da .csv fajl ne može da se učitava ukoliko je otvoren, stoga, ako pokrećete program I učitavate podatke iz njega, zatvorite prvo .csv fajl !

Regresija

Predstavlja nadgledano učenje, pošto modelu kažemo x a on nama treba da vrati y, pa čak i za one vrednosti za koje nema par (x,y).

Kada se završi proces obučavanja linearne regresije, znanje koje smo stekli se nalazi u **PRAVOJ**. Odnosno kada završimo proces obučavanja(fitovanja) mi smo izračunali **k i n** (prava je $y = k*x + n$).

Kada obučavamo neuronsku mrežu, imamo dve metode, **fit()** gde **obučavamo neuronsku mrežu** i **predict()** gde vršimo samo neku **predikciju**.

Kada vršimo predikciju mi za neki **NOVI** podatak, treba da predvidimo koliki će biti rezultat.

Ovde je x bio input podatak dok je y bio **LABELA**, zbog toga je regresija **nadgledano učenje**.

U ovakvom tipu zadatka(u većini slučajeva) na kolokvijumu nemamo potrebu za menjanjem modela, nego menjamo samo format inputa I outputa I tada formiramo listu svih inputa I outputa. Potom je

potrebno da pozovemo fit(odnosno istreniramo naš model) a nakon toga I da pozovemo predict() na nekih 2-3 primera, što bi predstavljalo čitav zadatak.

Uvod [nenadgledano učenje]

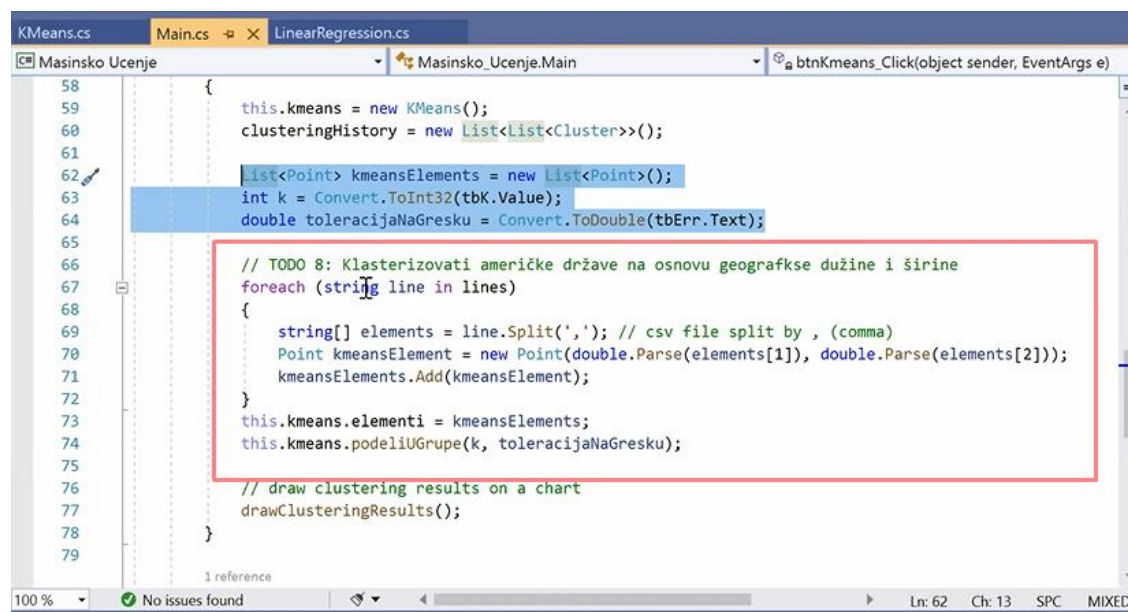
Predstavlja učenje bez učitelja, odnosno imamo input i model ali ne znamo šta model treba da dobije .

Ako se vratimo na primer sa podacima o pacijentu (krva zrnca, slika pluća i sl.) gde imamo neke labelirane podatke (imamo attribute) mi uz pomoć nenadgledanog učenja(klasterovanje, grupisanje) možemo grupisati(na osnovu rastojanja npr) elemente bez da znamo da li su imali Covid19 ili ne, I onda grupu gde su većina recimo imali Covid19, kažemo svi iz te grupe su imali Covid19. Što znači da na osnovu same sličnosti izmedju redova(entiteta)

K-means

Podelimo prostor(podatke) u k grupa, gledajući sredinu svake grupe kao kriterijum za određivanje novog centra, znači gledamo sredinu klastera kao opis klastera. Poenta je da algoritam zna koliko je k(koliko imamo grupa unapred)

Na kolokvijumu najčešće iz ovog tipa zadatka može doći kao TODO 8 iz vežbi 4, ono pre njega je algoritam I on se u suštini I ne menja, nego mi jedino što menjamo je ono oko modela , odnosno prilagođavamo naše podatke. Najčešće je potrebna transformacija input podataka.



```
58 {
59     this.kmeans = new KMeans();
60     clusteringHistory = new List<List<Cluster>>();
61     List<Point> kmeansElements = new List<Point>();
62     int k = Convert.ToInt32(tbK.Value);
63     double toleracijaNaGresku = Convert.ToDouble(tbErr.Text);
64
65
66     // TODO 8: Klasterizovati američke države na osnovu geografske dužine i širine
67     foreach (string line in lines)
68     {
69         string[] elements = line.Split(','); // csv file split by , (comma)
70         Point kmeansElement = new Point(double.Parse(elements[1]), double.Parse(elements[2]));
71         kmeansElements.Add(kmeansElement);
72     }
73     this.kmeans.elementi = kmeansElements;
74     this.kmeans.podeliUGrupe(k, toleracijaNaGresku);
75
76     // draw clustering results on a chart
77     drawClusteringResults();
78 }
79
```

Naivni Bayes

Nadgledano učenje, imamo primer inputa, primer output-a, i algoritam treba da nauči šta je na inputu važno odnosno šta nije važno I da na osnovu toga da neku predikciju.

Jako je bitno da od labele sa tekstom pretvorimo je u **labelu sa brojevima**, jer naši algoritmi rade s brojevima ! Najbitnije je da shvatimo kako vektorišemo tekst, odnosno kako od teksta pravimo broj.

U fit metodi formiramo znanje o našem modelu.

Voditi računa da mi vršimo aproksimaciju, pa zbir verovatnoća neće biti u sumi 1 kao što bi I trebalo, nego npr jedna će biti 0.7 jedna 0.9, nama je potrebno samo koja je veća I tu je kraj.

Na jednom od kolokvijuma je bilo da imamo više sentimenata, 0-negativan, 1-pozitivan, 2 – neutralan itd. Na drugom da imamo parove reči, pa da u predprocesiranju ishendlujemo kad dođe reč npr: *not good* mi zaključimo da je taj review(komentar/document) zapravo negativan a ne kao kad imamo obične reči pa bude pozitivan zbog reči *good*. Znači token ne mora da bude samo reč, može par reči biti jedan token.

Odnosno možemo koristiti neke engrame, bigrame (parove reči u rečenici , bi- je par od 2 reči). U ovom zadatku bi dovoljno bilo da u CountWords kada se od tokena formiraju mini rečnici samo grupišemo uzastopne reči. Pa onda u rečniku imamo { this movie : 1, movie was: 2, was good: 1, was bad: 10} gde je key taj engram a value broj njegovog pojavljivanja .

Na taj način možemo optimizovati ovaj algoritam (naravno ovaj rečnik će biti mnogo veći), odnosno tako hendlamo negaciju. Ako imamo npr :) ili neke slične emotikone, ako hoćemo to da hendlamo, to radimo u predprocesiranju (TextUtil) teksta, ili na kraju bustujemo nekim množenjem(brojem većim od 1 naravno) ako je u pitanju emotikon kako bi mu dali prednost.

Veštačke neuronske mreže

Nadgledani način učenja, imamo input i kažemo šta bi trebao da bude izlaz a sam model treba da nauči kako da namapira ulaz na izlaz. Poenta neuronske mreže je da nauči kako da izvrši mapiranje.

Mi imamo gomilu neurona koji su jako jednostavni i koji rade samo množenje inputa sa težinom, nakon toga ih saberimo i provučemo kroz aktivacionu funkciju i dobijemo izlaz. Ideja je samo da podesimo težine na pravi način, tako da za vector koji dobijamo na ulazu, dobijemo smislen output.

Pošto je u pitanju nadgledano učenje, i imamo labelu na izlazu, mi znamo šta bi trebao da bude rezultat a šta smo mi naprimer dobili, zbog toga koristimo i gledamo kolika je greška kako bi onda podešavali težine i dobili što manju grešku na izlazu.

Inicijalno random izgenerišemo težine kada krenemo sa procesom obučavanja a onda dovedemo input i output. Nakon toga imamo dva koraka, prvi je **forward() pass** gde bukvalno ulazni vector provučemo kroz sabirače/množače odnosno provučemo kroz neuronsku mrežu, potom je potrebno to provući i kroz recimo sigmoidnu **aktivacionu funkciju**. Onda gledamo koliko je svaka od tih težina dobrinela greški u izlaznom sloju i nakon toga težine samo modifikujemo (i to radimo iterativno).

Jedna iteracija: **forward pass** -> izračunamo *grešku* -> izračunamo *parcijalne izvode* -> **backward pass**

Mi u **backward** passu “štimo” težine koje su uticale na grešku.

Sigmoidna funkcija nam unosi nelinearnost u sistem.

Loss function() je funkcija greške koju izračunavamo posle forward pass-a. Obično koristimo [metodu najmanjih kvadrata](#). Mi izračunamo funkciju greške za svaku težinu da bi znali koju je potrebno “štimiti”.

Ukoliko težina dobrinosi da greška raste, težinu je potrebno smanjiti, ukoliko doprinosi da greška opada, onda tu težinu treba povećati.

Posle izračunavanje loss function moramo izračunati novu vrednost težine, za to koristimo **optimizer**, odnosno najčešće **algoritam gradijentnog pada (gradient descent)**. Zapravo optimizer koristi loss function kako bi odredio novu težinu. Kod GD je alfa koeficijent učenja, što je manji, obučavanje ide sporije ali jeftinije, mi nekad fiksiramo ali je u praksi često u praksi adaptivan.

Mi u **backward pass-u** zapravo *računamo gradijente* za svaku težinu, tj vidimo koliko je svaka težina uticala na grešku a potom koristimo **optimizer** (obično **gradient descent**) koji će nam modifikovati težine, pomoću ovog gradijenta koji smo upravo izračunali. **Learning reatom** (alfa) podešavamo kojim ćemo to intezitetom podešavati.

Epoha nam predstavlja **jedan prolaz** kroz **čitav data set** (ako imamo 100.000 redova, jedna epoha je prolazak kroz svih 100.000 redova).

Obično je optimalno koristiti **mini-batch**, odnosno “štimate” težine tek na nekoliko redova. Ne odmah posle svakog reda ali ne i tek na kraju, nego posle nekoliko redova, onda uradimo “štimanje” težina.

Normalizacija je process u kome mi attribute koji nisu u opsegu/domenu ostalih atributa, uzimamo i svodimo na opseg ostalih atributa. Obično vršimo normalizaciju da svi atributi budu u intervalu između 0 i 1.

Bias nam služi da transliramo aktivacionu funkciju, odnosno on nas dovodi blizu “sweat spot”-a (ako nam aktivaciona f-ja da izlaz od 1000 a bias je -900, mi ćemo se translirati mnogo bliže “sweat spot”u zbog nje). Njegova težina se samo menja a input ne i koristi se za translaciju aktivacione funkcije.

Obično mi na kolokvijumu/projektu ne moramo implementirati ovaj kod s vežbi, jer ćemo ga već imati, pa je potrebno samo da je koristimo kao black-box.

Nama na kolokvijumu bude zadatak da zapravo uradimo normalizaciju i transformaciju atributa, jer nema smisla da implementiramo neuronsku mrežu iz početka nego transformišemo podatke kako bi izvukli maksimum iz modela.

Može na kolokvijumu biti dato da se doda aktivaciona neka funkcija po formuli, tipa tanges hiperbolični i slično.

Obično ono što menjamo na kolokvijumu se nalazi pre svega u klasi Program.cs, ono što se od nas traži je da napravimo inpute i outpute za neuronsku mrežu i da pozovemo fit i predict. Obično je isti kostur za ovaj tip zadatka, jedino je drugačiji data-set.

Poenta kolokvijuma je da isprocesiramo podatke, transformisemo ih i da napravimo neku predikciju.

Priprema 2019/2020 – Game of Thrones

Laki:

U regresiji samo ubacimo X, ubacimo Y i pozovemo fit.

Srednji:

Proverimo da li je potrebna neka transformacija.

Voditi računa da naš K-means radi samo sa 2D tačkama (imali smo Point() u 2D), zbog toga moramo prošiti Point klasu na 5 tačaka jer imamo 5 atributa zahtevanih u zadatku (book1, book2, book3, book4, book5) I da promenimo funkciju rastojanja (da sada bude $\sqrt{x_1^2 + y_1^2 + z_1^2 + \dots} - \sqrt{x_2^2 + y_2^2 + z_2^2 + \dots}$) I tjt)

Ako imamo u labeli(atributu) neke redove sa missing value odnosno, null ili empty, moramo te vrednosti popuniti ili će program pucati.

Teški

One hot vector je vector koji nam pomaže kod transformacije teksta u broj, ideja:

Napravimo vector dužine koliko imamo različitih vrednosti u problematičnoj labeli(atribut sa tekстом), potom svakoj jedinstvenoj vrednosti odredimo **one hot** mesto u vektoru na kojem će samo ona biti. Npr: [0 0 0 1] označava Srbiju, [0 0 1 0] označava Crnu Goru itd itd, bitno je da svaka jedinstvena vrednost ima svoje **one hot** mesto u vektoru.

Najbitnije je voditi računa da odradimo **normalizaciju** i **transformaciju** ukoliko imamo neke **kategoričke** attribute koje koristimo !

Ako dobijemo dosta male vrednosti u nekom atributu tipa od 0.01 do 0.1, njih takođe moramo normalizovati, odnosno dići na viši nivo !

Pošto nisu problem koje mi rešavamo komplikovani, dovoljno je 2-3 sloja neuronske mreže, ali da imamo više outputa, onda već treba razmisliti o dodavanju slojeva. I na našem nivou to ide intuitivno, ali radi se i eksplorativna analiza podataka gde analiziramo neke parametre i određujemo broj slojeva mreža I slično.

Sa povećanjem broja slojeva, možemo doći do overfitovanja, pa defakto ne treba preterivati u dodavanju slojeva.

Sigurno će biti u zadatku da izračunamo kolika je tačnost.

Kod K-means-a treba napraviti samo generičku funkciju rastojanja I onda na kolokvijumu samo ubacimo koja je dimenzija I tjt. I metodu lakta (elbow metoda) da napravimo, kako bi na pametan način izabrali k I tako optimizovali naše rešenje(alice ako lakat metoda bude, onda je to baš težak zadatak, ali male su šanse za nju).