Custom Vision i LUIS servisi na Microsoft Azureu

Markovinović, Eugen

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Department of Mathematics / Sveučilište Josipa Jurja Strossmayera u Osijeku, Odjel za matematiku**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:126:394701

Sveučilište J.J. Strossmayera u Osijeku

Odjel za matematiku

Eugen Markovinović

Custom Vision i LUIS servisi na Microsoft Azureu

Diplomski rad

Osijek, 2018.

Sveučilište J.J. Strossmayera u Osijeku

Odjel za matematiku

Eugen Markovinović

Custom Vision i LUIS servisi na Microsoft Azureu

Diplomski rad

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

Osijek, 2018.

Sadržaj

1 Uvod

Značajnim napretkom tehnologije te pogotovo napretkom računala i njihove snage, pojavili su se novi problemi, ali i nova rješenja u svim područjima znanosti. Problemi su postali ili presloženi ili preveliki obujmom da bi ih čovjek mogao sam riješiti. Nije bilo moguće riješiti probleme na klasične načine, te se javila izrazita potreba da algoritmi imaju gotovo ljudsko razmišljanje. Upravo iz toga razloga došlo je do razvoja umjetne inteligencije, te zajedno s njom i koncepta strojnoga učenja.

Upravo razvojem strojnoga učenja, postalo je moguće rješavati raznovrsnu količinu problema. Neki od problema na koje se primjenjuju tehnike strojnoga učenja do sada su filtriranje spam mailova, predviđanje vremena, prepoznavanje lica, osoba i objekata sa slika, prepoznavanje govora... Inženjeri, računarci, fizičari, matematičari, biolozi te brojni drugi već pokušavaju primijeniti tehnike strojnoga učenja na svoje probleme, te htjeli mi to ili ne, naša budućnost je vjerojatno usko povezana sa razvojem strojnoga učenja.

Ovaj rad će objasniti ukratko osnovne koncepte i ideje strojnoga učenja. Rad će opisati Microsoft servise LUIS i Custom Vision, te prikazati izradu bota korištenjem spomenutih servisa. Biti će pokazana moguća implementacija korištenjem bot frameworka u Visual Studiu, te kao konačan produkt praktičnog dijela rada će biti funkcionalan bot koji vrši jednostavnu komunikaciju sa korisnikom te ima sposobnost prepoznavanja konveksnih mnogokuta sa slika.

Ideja ovoga rada je približiti koncept strojnoga učenja čitatelju, te pokazati na jednostavnom primjeru kako napraviti funkcionalnu primjenu strojnoga učenja na način koji je dostupan i onima koji znaju samo osnove strojnoga učenja.

2 Strojno učenje

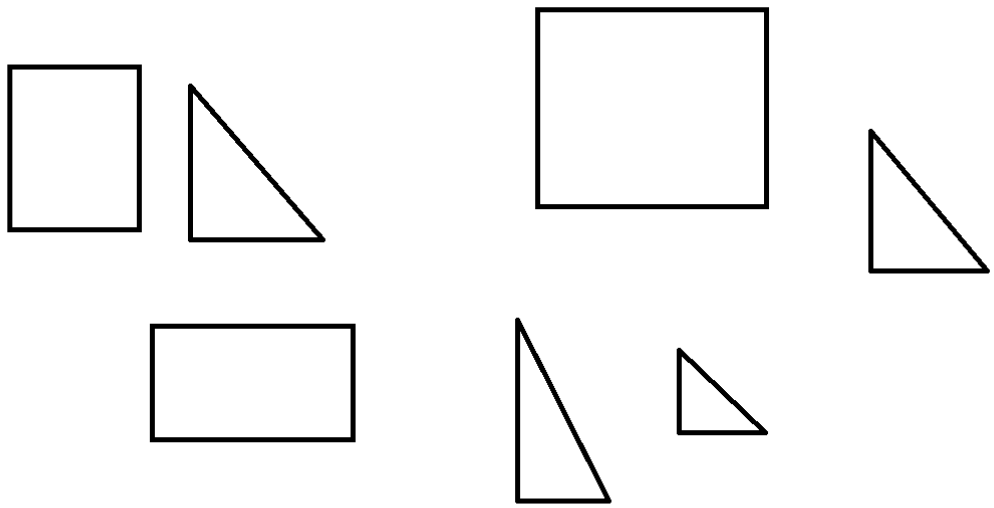
2.1 Osnovni pojmovi

Već u uvodu je spomenuto kako primijeniti strojno učenje, no zapravo nikako nije objašnjeno što je to. Postoje brojne definicije te se općenito pod ovim pojmom misli na bilo koji način pod kojim se stroj, na temelju podataka koje posjeduje, poboljšava i obavlja svoj zadatak bolje. Malo općenitiju definiciju imamo za algoritam strojnoga učenja.

Definicija 2.1. (Mitchell 1997.) *Računalni program uči iz iskustva E na klasi zadataka T gdje mu performanse mjerimo sa P, ukoliko mu se performanse na zadatku T mjerene sa P poboljšavaju sa iskustvom E.*

Zadaci T mogu biti raznovrsni, prema [1] najčešći su:

- Klasifikacija – algoritam treba odrediti kojoj od k kategorija određeni ulaz pripada, neki od primjera klasifikacije su prepoznavanje određenih objekata, određivanje što je spam mail a što ne. Možemo pokušati razvrstati likove na trokute i pravokutnike (Slika 2.1).
  Upravo u praktičnome dijelu ovoga rada je obrađen jedan od problema klasifikacije, prepoznavanje konveksnih od nekonveksnih mnogokuta.

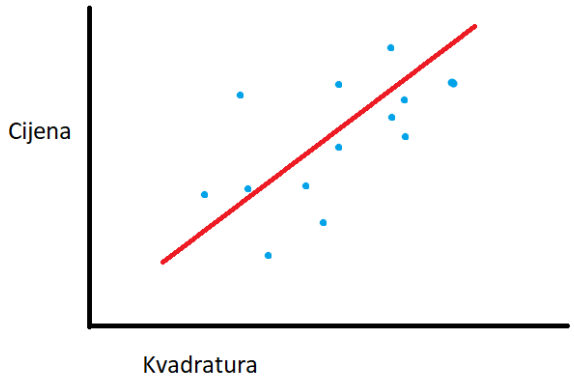


Slika 2.1

- Klasifikacija s nedostajućim ulazima – Ovaj zadatak je identičan kao prethodni zadatak klasifikacije uz pretpostavku da su nam ulazi potpuni. No kako se često

javlja u primjenama, mjerenja nekada nedostaju, dio slike je zaklonjen nasumičnim prolaznikom. Ovaj problem tada postaje znatno teži.

- Regresija – U ovome tipu problema od programa se traži da kao rezultat na ulaze izbaci realni broj. Ovo je primjerice korisno kod predviđanja cijena stanova, gdje kao ulaz imamo kvadraturu, a kao izlaz cijenu stana.



Slika 2.2

- Transkripcija – Prepoznavanje znakova sa slike
- Prevođenje – Prevođenje prirodnog jezika
- Prepoznavanje anomalija – U podatcima pronaći one koji se ističu, ne odgovaraju ostalima ili ne bi smjeli biti takvi kakvi jesu

Za mjerenje P se često uzima broj pogrešno obrađenih primjera te se mjeri točnost i razina pogreške na razne načine. Primjerice kod klasifikacije možemo mjeriti broj onih koji su dobro svrstani u svoju kategoriju, te broj onih koji nisu dobro svrstani.

Uvesti ćemo sljedeće oznake za klasifikaciju koje se odnose na jednu kategoriju:

TP – broj dobro svrstanih u kategoriju

FP – broj onih koje je algoritam svrstao u kategoriju, a zapravo ne pripadaju u nju

FN – broj onih koje algoritam nije svrstao u kategoriju, a trebao je

TN – broj onih koje algoritam nije svrstao u kategoriju, a nisu ni trebali biti u njoj

Zatim možemo mjeriti preciznost te takozvani opoziv. **Preciznost** ćemo definirati kao:

$$\frac{TP}{TP + FP}$$

a **opoziv** kao:

$$\frac{TP}{TP + FN}$$

Preciznost nam govori postotak koliko od onih koje svrstavamo u kategoriju, dobro svrstavamo. Opoziv nam govori koliko od ukupnih koji bi trebali biti svrstani u kategoriju, mi svrstamo u nju. Koji od ova dva je bitniji ovisi od našega zadatka i prirode problema. Primjerice, ako detektiramo karcinom kod osoba, želimo imati što veći opoziv, jer ni u kojem slučaju ne želimo nekome reći da nema karcinom, a zapravo ga ima. S druge strane, ako beremo gljive, i biramo jesu li otrovne ili ne, želimo imati preciznost što veću, jer iako možda nećemo ubrati sve gljive koje su u šumi, nikako ne želimo ubrati otrovnu.

Često se koriste dodatni podatci kako bi se testirale performanse našega algoritma, skup koji čine podatci na kojima testiramo algoritam ćemo zvati **test skup**.

Zadnji i najbitniji dio algoritma strojnog učenja je iskustvo E. Iskustvo algoritam dobiva na podatcima koje mu proslijedimo. Ovaj skup podataka se može zvati i **trening skup** jer mi zapravo „treniramo“ algoritam kako bi postao bolji.

Nositelji informacija u podatcima trening skupa su takozvane **komponente učenja** (eng. features). Ako se vratimo na primjer procjene cijene stanova, naš ulaz ne mora nužno biti samo kvadratura, on može sadržavati i broj soba, veličinu dvorišta, boju kuće... Sve ove značajke mogu utjecati na cijenu, te su sve one komponente učenja. Postaje jasno da izbor ovih komponenti uopće nije trivijalna stvar.

Ovisno o iskustvu koje algoritam posjeduje razlikujemo :

- Učenje s nadzorom – učenje u kojemu algoritam ima uvid u podatke trening skupa, te kako bi oni trebali biti obrađeni
- Učenje bez nadzora – algoritam ne zna kakvi podatci na trening skupu moraju biti, te sam odlučuje što s njima.
- Podržano učenje – algoritam vrši interakciju s okolinom, te tako sam generira svoj trening skup. Ovaj tip učenja je specifičan za robote, igre i bilo kakav sustav gdje algoritam ima mogućnost sam istražiti kakav mu je podatak.

2.2 Učenje s nadzorom

2.2.1 Linearna regresija

Iako postoje brojni algoritmi koji uče s nadzorom, najjednostavniji ali vrlo lagan za reprezentirati je linearna regresija. Linearna regresija, kako joj i ime kaže, rješava problem regresije, odnosno za ulaz $x \in \mathbb{R}^n$ vraća predviđanje $\hat{y}$ koje želimo da bude što bliže y. Naš x je ovdje upravo sastavljen od komponenti učenja. Linearna regresija računa $\hat{y}$ kao: $\hat{y} = \omega^\intercal x$, gdje je $\omega \in \mathbb{R}^n$. ω možemo promatrati kao vektor težina, gdje svaka težina određuje koliko je pojedina komponenta učenja utjecajna. Ako iz trening skupa imamo podatke kojima su izmjerene komponente učenja i odgovarajuća rješenja, jedan podatak iz trening skupa izgleda kao (x^i, y^i), gdje je $i = 1 \ldots . m$, m broj podataka u trening skupu. Onda mi želimo da za $\forall (x^i, y^i)$ vrijedi $y^i \approx \omega^\intercal x^i = \hat{y}^i$. Ukupnu pogrešku možemo promatrati kao $\sum_{i=1}^{m} \left(\hat{y}^i - y^i\right)^2$. Minimiziranjem ovoga problema dolazimo do težina ω. Primijetimo kako imamo sumu m komponenti, te ako je m izrazito velik, ovo postaje vrlo zahtjevan problem. Minimizaciju vršimo nekim od algoritama za minimizaciju, najčešće gradijentnom metodom. Ukoliko je m prevelik, zna se uzimati samo dio podataka koji se onda koriste za minimizaciju. Inače, funkcija koja predviđa $\hat{y}$ se naziva **funkcija hipoteze**. Za linearnu regresiju je hipoteza $h(x) = \omega^\intercal x^i$.

2.2.2 Logistička regresija

Postoji način da istu metodu rješavanja problema regresije primijenimo na problem klasifikacije. Ako primjerice imamo samo dvije kategorije koje moramo klasificirati, zamijenimo funkciju hipoteze iz linearne regresije sa novom funkcijom $h(x) = \frac{1}{1+e^{-\omega^\intercal x}}$ primijenili smo takozvanu sigmoid funkciju koja daje rezultate između 0 i 1.

Sada klasificiramo primjere iz trening skupa tako da ukoliko $h(x)$ daje rezultate bliže 0, svrstavamo ga u prvu kategoriju, a inače u drugu.

Zanimljivo je da iako rješavamo problem klasifikacije, ova metoda ima u nazivu regresija.

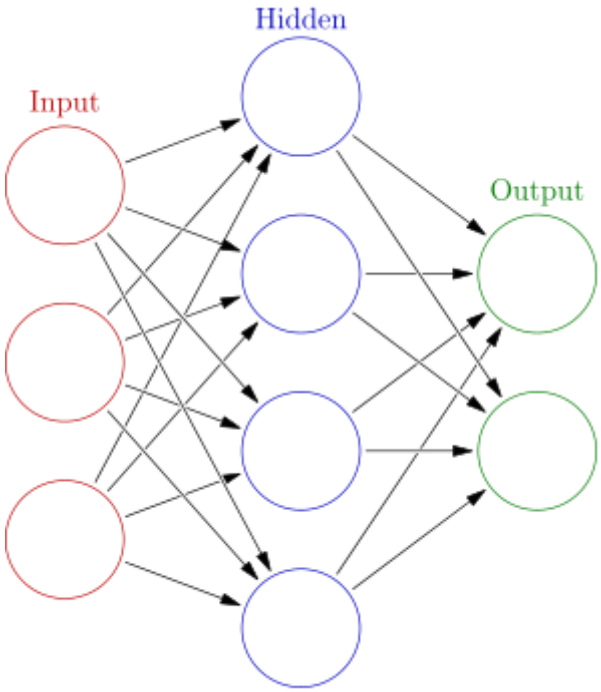
Postoje i drugi algoritmi koji su uspješniji od navedenih, primjerice support vector machine (SVM), decision tree....

2.2.3 Neuronske mreže

U uvodu je spomenuto kako se snaga računala znatno povećala te je dovela do novih rješenja. Upravo glavni razlog velikog uspona strojnoga učenja u zadnje vrijeme su neuronske mreže. Iako su već starija ideja, dobivena pokušajem imitacije rada mozga i

imitacije neurona u mozgu, tek nedavno su započele velike implementacije i primjene. Neuronskim mrežama se rješavaju gotovo svi moderni problemi strojnoga učenja, od prepoznavanja sa slika do prevođenja.

Neuronska mreža se sastoji od više neurona (čvorova) koji su organizirani u slojeve. Prvi sloj, ili ulazni sloj, sadrži ulaze, dok izlazni sloj daje rezultat. Između se nalaze skriveni slojevi (Vidi Slika 2.3).



Slika 2.3

Ako nam je $g(x) = \frac{1}{1+e^{-\omega^\intercal x}}$ sigmoid funkcija a čvorove neuronske mreže označimo sa $a_i^{(j)}$ gdje je eksponent broj sloja od lijevo prema desno, a indeks broj čvora od gore prema dolje. Tada imamo $a_i^{(j)} = g(\Omega_{i,1}^{(j-1)} a_1^{j-1} + \Omega_{i,2}^{(j-1)} a_2^{j-1} + \cdots + \Omega_{i,n}^{(j-1)} a_n^{j-1})$, sada je jasno da je $\Omega^{(j)}$ matrica između sloja j i $j+1$. Ovo znači da svaki sloj ima svoju matricu $\Omega^{(j)}$ čije dimenzije ovise o slojevima u kojima se nalazi.

Primijetimo da ovakvim dizajnom modela mi zapravo sakrivamo parametarski oblik funkcije hipoteze iza slojeva neuronske mreže. Neuronska mreža je sposobna sama odrediti svoje apstraktne komponente učenja, bez da ih mi moramo određivati!

No kako uopće trenirati neuronsku mrežu? Njezina kompleksna struktura znatno otežava standardni način. Krajem prošloga stoljeća došlo se do backpropagation algoritma. On idući unatrag po slojevima mijenja težine distribuirajući greške po čvorovima. Više o njemu se može naći u [1].

Daljnjim radom na neuronskim mrežama došlo se do dubokih mreža, koje su početak deep learninga ili dubokog učenja. Detaljniji opis dubokoga učenja nije u fokusu

ovoga rada, te ukoliko je čitatelj zainteresiran može više pronaći u [1]. Duboko učenje neće biti dalje obrađivano, iako metode korištene u praktičnome dijelu ovoga rada koriste modele iz dubokoga učenja.

2.3 Učenje bez nadzora

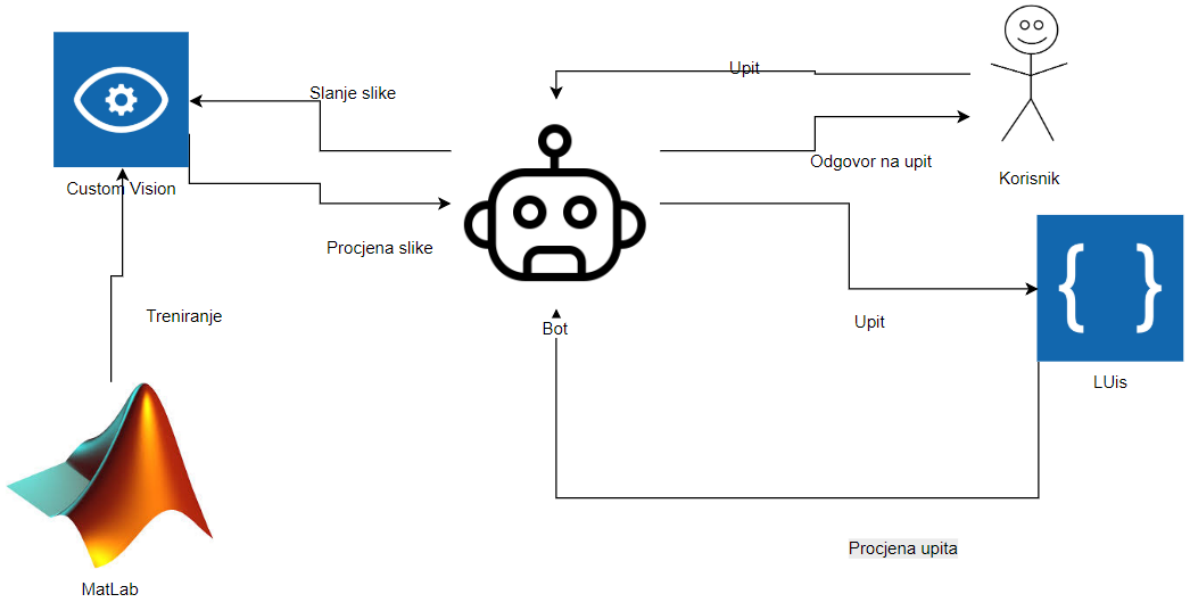
Učenje bez nadzora spada u posebnu klasu problema, jer algoritam na početku nema nikakve upute za rješavanje zadatka osim osnovnih podataka i zadatka koji mora obaviti. Zasigurno najistaknutiji problem iz ovoga područja je klasteriranje podataka, podatci se pokušavaju podijeliti u disjunktne klastere gdje po nekoj logici najbolje odgovaraju. Termin najbolji se često definira kao onaj s najmanjom udaljenosti.

Najpoznatiji algoritam, ali ne i jedini, koji rješava ovaj problem je k-means algoritam. Ukratko, algoritam kreće od nasumičnog skupa reprezentanata, te računa klastere oko svakoga od njih na temelju udaljenosti. Zatim računa najboljeg reprezentanta svakoga klastera, te ponovno računa nove klastere. Tako ponavlja dokle god može poboljšati rezultat.

Posebni problemi koji spadaju u učenje bez nadzora su detekcija anomalija, principal component analysis(PCA)...

3 Alati i servisi za izradu praktičnog dijela

Praktični dio diplomskog rada obuhvaća izradu jednostavnoga chat bota te prepoznavanje konveksnosti mnogokuta sa slike. Putem bota korisnik komunicira sa povezanim servisima te uploada svoje slike, koje se onda testiraju na Azure Custom Vision servisu, te bot dostavlja odgovor. Praktični dio obuhvaća i treniranje Custom Vision servisa i izradu podataka za treniranje. Slika 3.1 prikazuje osnovni model praktičnog dijela.



Slika 3.1

3.1 Alati korišteni za izradu

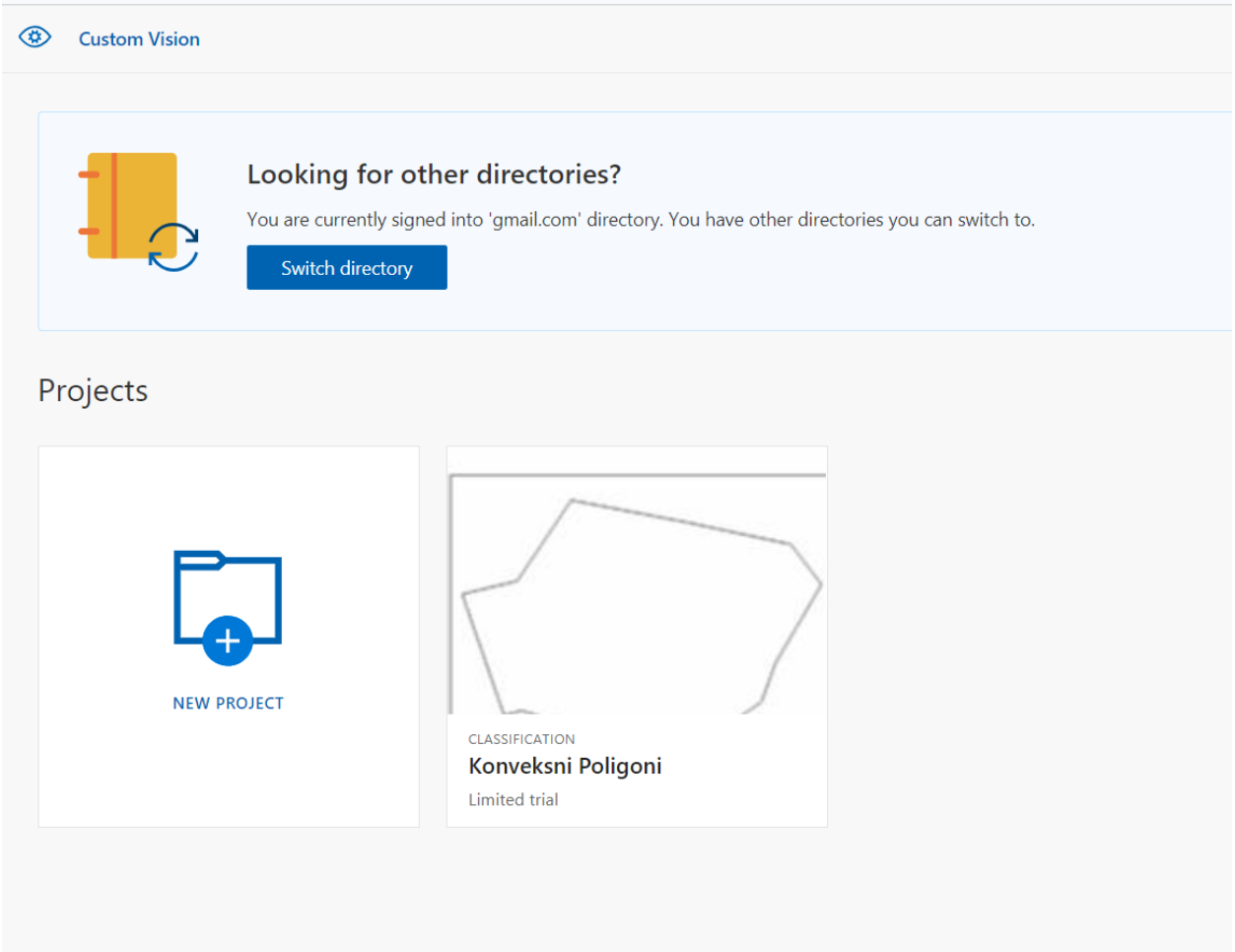
Za izradu je korišten Visual Studio Community Edition, matlab, Azure platforma sa servisima LUIS i Custom Vision. Programski jezik korišten je c# i .NET framework.

- Visual Studio je integrirano razvojno okruženje sa brojnim featurima i mogućnostima koje uključuju dodavanje raznih paketa, pisanje koda u različitim programskim jezicima
- Matlab je matematički alat koji služi za matematičke izračune, crtanje slika...

- Microsoft Azure je stalno rastuća platforma servisa koji su dostupni svima u oblaku. Servisi uključuju različite stvari, ne samo za strojno učenje nego i primjerice virtualne mašine, spremišta za baze podataka...

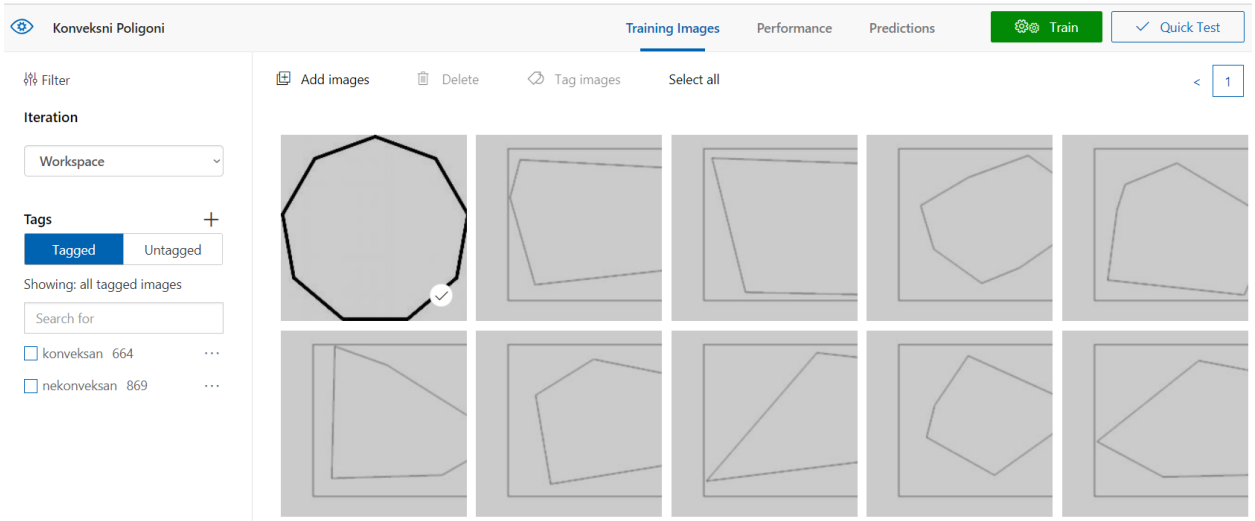
3.2 Custom Vision

Custom Vision je servis koji omogućava vrlo jednostavnu i intuitivnu izradu modela za prepoznavanje i klasifikaciju slika. Iako se naplaćuje za veće upotrebe, postoji besplatna verzija. Sve što je potrebno je kreirati projekt (Slika 3.2) te krenuti uploadati slike na njega. Na sve slike moramo staviti tagove, preko kojih će CustomVision učiti.



Slika 3.2

Zatim odaberemo prave postavke te jednostavno kliknemo na train i samo tako je Custom Vision spreman za upotrebu.



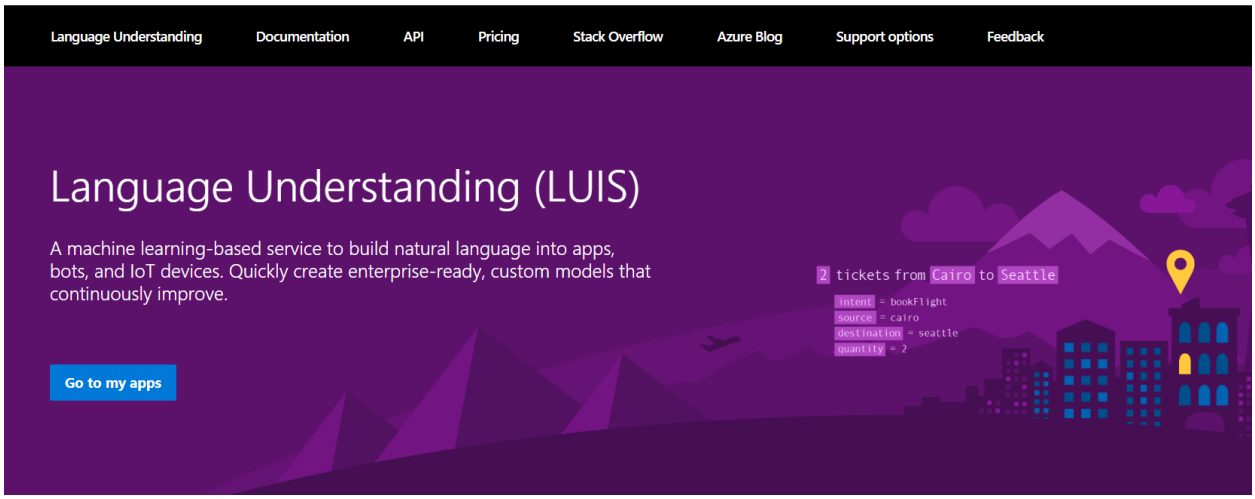
Slika 3.3

Treniranjem modela kreiramo novu iteraciju, pa možemo uspoređivati sa prethodnim treninzima, te vidjeti poboljšanja.

Kada imamo istrenirani model, možemo uvijek poslati novi primjer na procjenu. Custom Vision onda vraća tagove i vjerojatnosti za svaki tag, te time daje procjenu za našu sliku.

3.3 LUIS

LUIS ili Language understanding je također servis na kojemu možemo istrenirati model. Glavna svrha LUIS-a je prepoznavanje jezika te određivanje točnih namjera onoga tko je napisao tekst. Isto kao i Custom Vision, naplaćuje se po broju upita, ali ima i besplatnu verziju za nekomercijalne upotrebe.



Slika 3.4

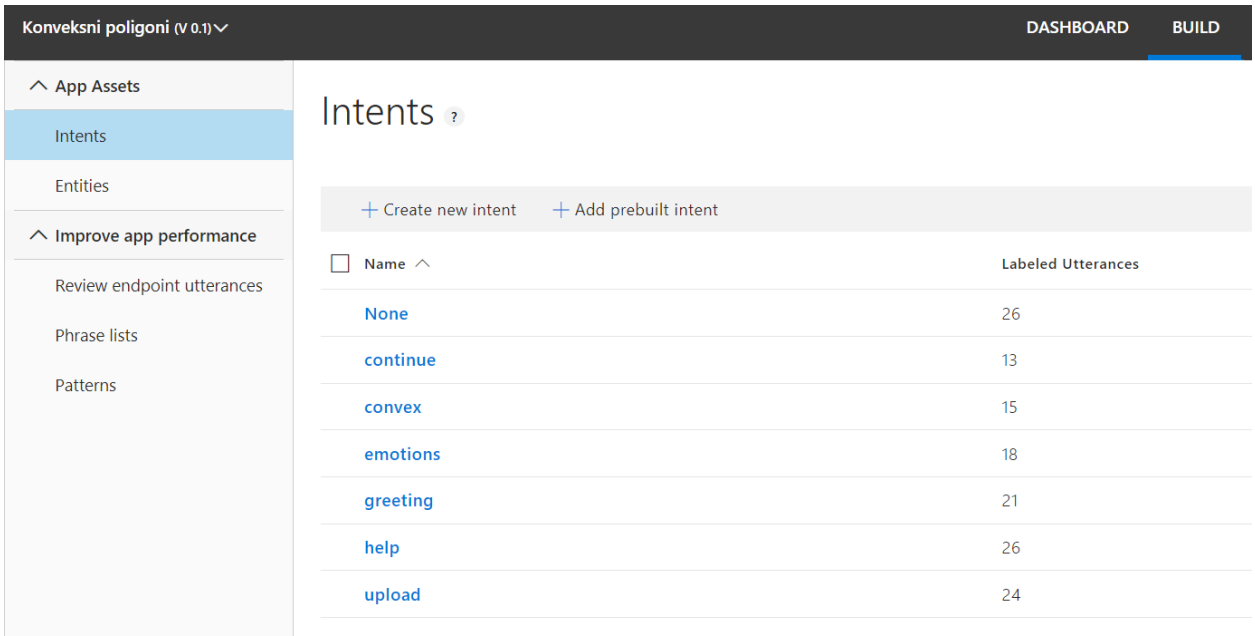
Kao i kod Custom Vision-a jednostavno kreiramo novu aplikaciju (Vidi Slika 3.5).

Slika 3.5

Nakon kreiranja aplikacije potrebno je sastaviti naš model. Dodajemo namjere (eng. intents) koje želimo da LUIS prepoznaje, te za svaku namjeru moramo dodati izjave koje se vežu uz nju. Primjerice ako procijenimo da je namjera naručivanje avionske karte, dodati ćemo izjave kao :

- Htio bih naručiti kartu
- Trebam kartu za let do Moskve
- Želim biti na letu do Moskve
- Tri karte molim

Što više izjava damo LUIS-u to je veća vjerojatnost da će on prepoznati korisnika kada bude htio naručiti kartu.



Slika 3.6

Kada kreiramo namjere, postoji i opcija za kreiranje entiteta, koji služe za dodatno razumijevanje. U našemu primjeru gdje je namjera naručivanje avionske karte bi mogli

dodati entitet grad. Onda označimo u našim izjavama svuda gdje se pojavljuje grad te će LUIS naučiti kako prepoznavati gdje putnici putuju.

Zadnji korak je treniranje našega modela, LUIS će se pobrinuti za sve ostalo. Naš model je već spreman za korištenje, no uvijek je dobro dodavati i uređivati izjave kako bi i sami utjecali na daljnje učenje LUIS-a.

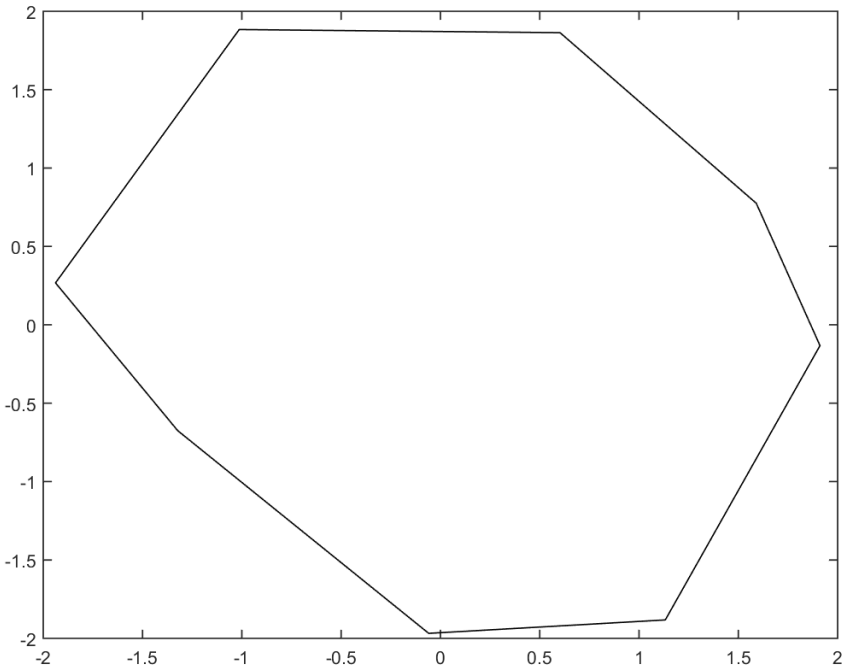
4 Izrada praktičnog dijela

Već je opisan model i alati korišteni, glavni izazov implementacije je bio funkcionalno povezivanje i komunikacija sa servisima.

4.1 Izrada trening skupa za Custom Vision

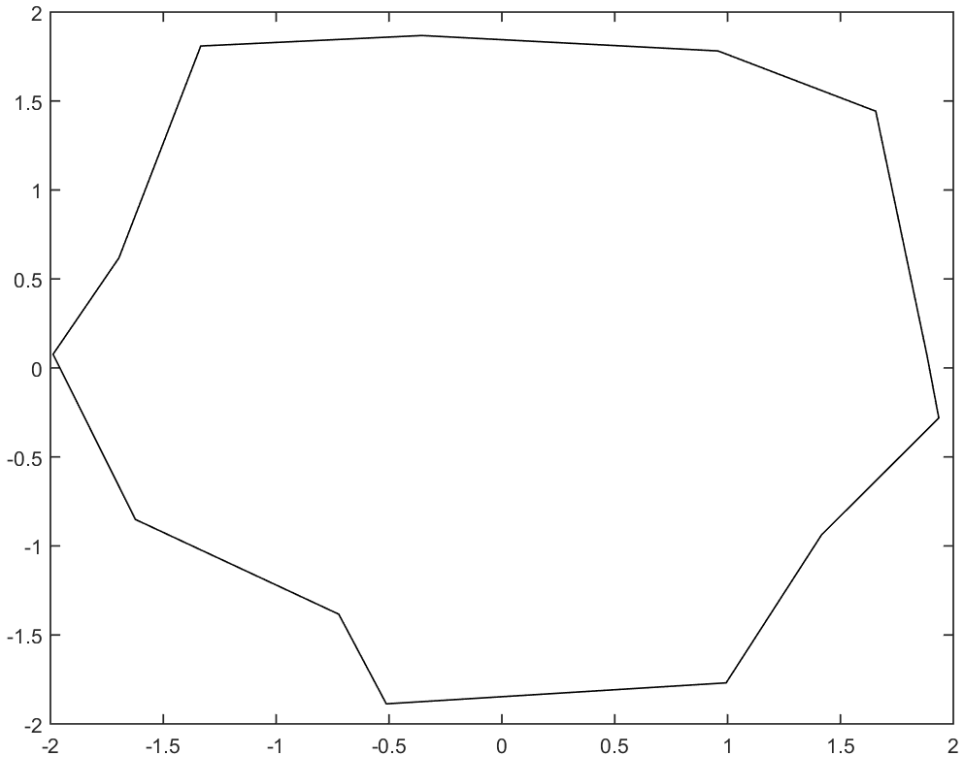
Ideja je bila izraditi servis koji će moći prepoznati sa slike konveksni mnogokut. Razlog zašto je odabran ovaj problem je jer je konveksne mnogokute relativno jednostavno generirati te ih je lako okarakterizirati. (Slika 4.1)

Definicija 4.1. *Mnogokut je konveksan u dvije dimenzije ukoliko dužina između svake njegove dvije točke leži unutar samoga mnogokuta.*



Slika 4.1

Iz definicije je jasno kako konveksni mnogokut mora sadržavati i sve svoje dijagonale. Također vrijedi da kako bi mnogokut bio konveksan, svi unutarnji kutovi mu moraju biti manji od $180°$. Ovo se može pokazati tako da se jednostavno pokaže da kada je neki od unutarnjih kutova manji od $180°$, onda dijagonala između njegova dva susjeda ne leži unutar mnogokuta. (Nekonveksan mnogokut na Slika 4.2)



Slika 4.2

Za kreiranje slika korišten je matlab, a konveksni mnogokuti su kreirani na način da se odabere broj kutova n. Zatim se u smjeru obrnutom od kazaljke na satu nasumično stavljaju točke. (Pogledaj kod na Slika 4.3)

```
% angle of the unit circle in radians
circleAng = 2*pi;
% the average angular separation between points in a unit circle
angleSeparation = circleAng/numVert;
% create the matrix of angles for equal separation of points
angleMatrix = 0: angleSeparation: circleAng;
% drop the final angle since 2Pi = 0
angleMatrix(end) = [];
% Specify polygon variables
radius = 1;
radVar = 1; % variance in the spikiness of vertices
angVar = 1; % variance in spacing of vertices around the unit circle
% generate the points x and y
for k = 1:numVert
    x(k) = (radius + radius*rand(1)*radVar) * cos(angleMatrix(k) + angleSeparation*rand(1)*angVar);
    y(k) = (radius + radius*rand(1)*radVar) * sin(angleMatrix(k) + angleSeparation*rand(1)*angVar);
end
```

Slika 4.3

Kako je Custom Vision-u odmah potrebno poslati i tagove slike, moramo odmah provjeriti konveksnost mnogokuta koji šaljemo. To radimo provjerom unutanjih kuteva,

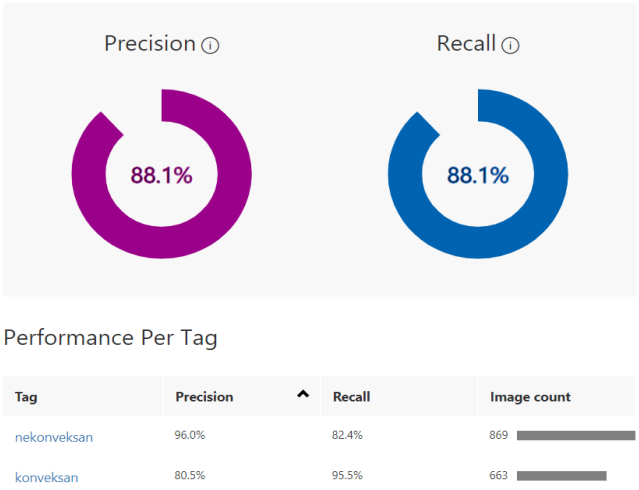
te zatim slike spremamo u dva odvojena foldera, ovisno jesu li mnogokuti konveksni ili ne. (Provjera kutova u kodu na Slika 4.4)

```
v1 = [px(1) - px(end), py(1) - py(end)];
v2 = [px(2) - px(1), py(2) - py(1)];
signPoly = sign(det([v1; v2]));
for k = 2:numPoints-1
    v1 = v2;
    v2 = [px(k+1) - px(k), py(k+1) - py(k)];
    curr_signPoly = sign(det([v1; v2]));
    % check that the signs match
    if not (isequal(curr_signPoly, signPoly))
        isConvex = false;
        return
    end
end
% check the last vectors
v1 = v2;
v2 = [px(1) - px(end), py(1) - py(end)];
curr_signPoly = sign(det([v1; v2]));
if not (isequal(curr_signPoly, signPoly))
    isConvex = false;
else
    isConvex = true;
end
end
```

Slika 4.4

Kada imamo slike razvrstane po konveksnosti u foldere, sve što trebamo napraviti je uzeti ih i poslati Custom Vision-u. Jednostavno korištenjem Visual Studia i NuGet paketa Microsoft.Cognitive.CustomVision.Training, stvorimo konekciju s našim Custom Vision servisom, napravimo tagove „konveksni“ i „nekonveksni“ te mu šaljemo iz foldera u kojima su spremljene, odgovarajuće slike s odgovarajućim tagom.

Sve što tada trebamo je istrenirati poslane slike i imamo prvi model za prepoznavanje. Nakon što istreniramo model dobre je provjeriti performanse, konkretno u ovome slučaju vidi Slika 4.5

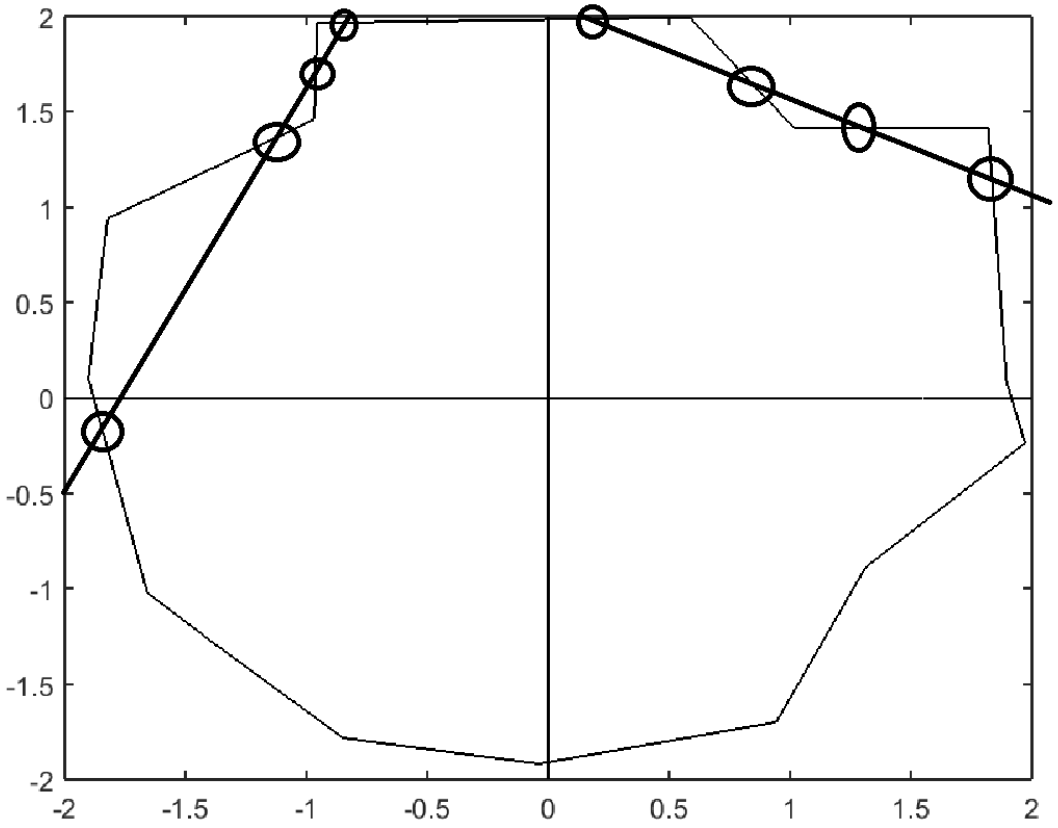


Slika 4.5

Trebalo bi napomenuti da ovakav način kreiranja slika ima velike mane, jer Custom Vision neće reagirati dobro na slike izvana, koje nisu generirane matlab-om. Naravno moguće je dodati umjetne smetnje na slike, no zasada je ovaj primjer samo radi demonstracije.

Kreiranje velikog i raznovrsnog trening skupa je uvijek u interesu povećanja kvalitete modela, te je često trening skup, ako je dovoljno dobar, najbitniji dio izrade procjenitelja.

Custom Vision je zamišljen za drugačije tipove problema, sa manjom varijabilnošću. Konveksni i nekonveksni mnogokuti ga izrazito muče jer dolaze u raznolikim oblicima. Konkretno, za klasificiranje slika konveksnih i nekonveksnih mnogokuta vjerojatno bi bilo bolje primijeniti neku od metoda klasične analize slike, bez strojnoga učenja. Primjerice mogli bi povlačiti ravne linije preko slike, te ako neka od linija siječe mnogokut u 3 ili više točaka, onda mora biti nekonveksan. (Vidi Slika 4.6)

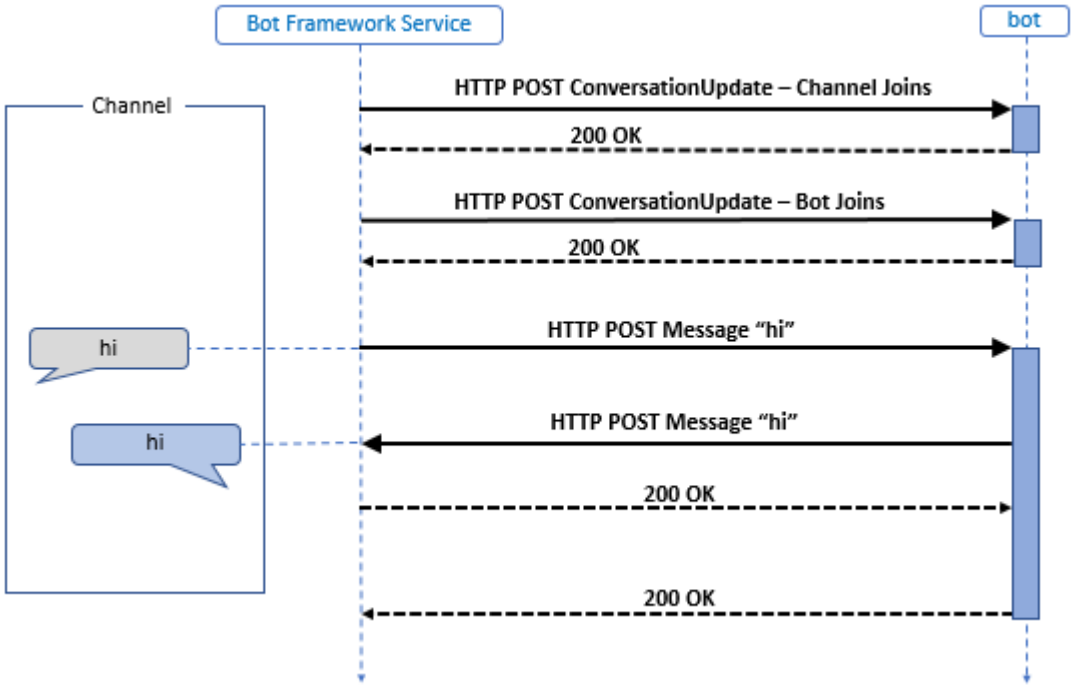


Slika 4.6

4.2 Izrada bota

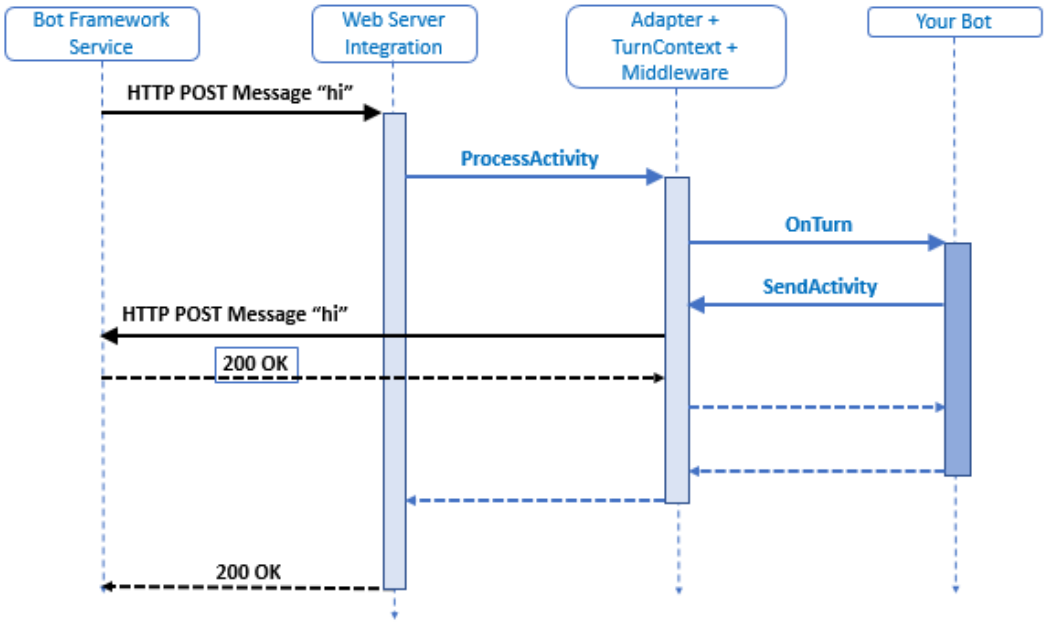
Bot je općenito dio programa zadužen za komunikaciju sa korisnikom, te bi trebao ostaviti dojam komunikacije slične komunikaciji između dva čovjeka. Naravno, postoje jednostavniji i složeniji botovi, no svrha svakoga od njih je interakcija sa korisnikom.

Bot koji je napravljen u praktičnome dijelu ovoga rada je napravljen isključivo u Visual Studiu uz pomoć bot framework-a i LUIS-a. LUIS je već objašnjen, a bot framework je skup korisnih alata i paketa napravljenih isključivo za izradu bota. On kreira servis na koji se korisnik spaja te tako kreće komunikacija.



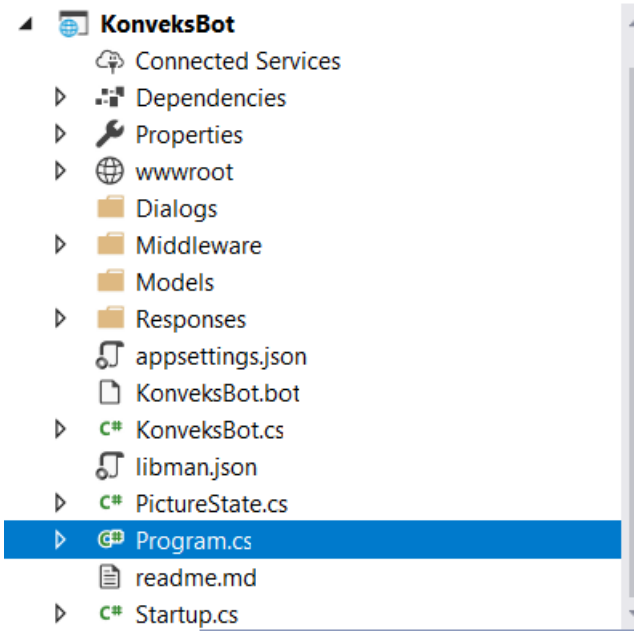
Slika 4.7

Sve aktivnosti i poruke koje prolaze se šalju HTTP protokolom u JSON obliku, koje adapter onda pretvara u takozvani TurnContext i zove middleware. Prije nego dođu do bota, poruke prolaze middleware sloj. Middleware je sloj koji procesira, predobrađuje i priprema poruke za bota. Ovaj sloj nije nužan, ali je često vrlo koristan. Sam web server je integriran u APS.NET-u, te se jednostavno izgradi sa build naredbom.



Slika 4.8

Izgled rješenja za bota se sastoji od implementacije middleware-a, dijaloga, odgovora, samoga bota i modela koje bot koristi. Modeli služe sa dodatne implementacije i rad primjerice sa slikama.



Slika 4.9

Startup se pokreće na samome početku, te on implementira, postavlja middleware i osnovne konfiguracije bot-a.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddBot<KonveksBot>(options =>
    {
        options.CredentialProvider = new ConfigurationCredentialProvider(Configuration);
        var middleware = options.Middleware;

        IStorage dataStore = new MemoryStorage();

        //Add middleware below
        middleware.Add(new UserState<UserData>(dataStore));
        middleware.Add(new ConversationState<ConversationInfo>(dataStore));
        middleware.Add(new CosmosMiddleware());
        // Add Regex ability below
        middleware.Add(new RegExpRecognizerMiddleware()
        .AddIntent("upload", new Regex("upload picture(?:s)*(.*)|upload pic(?:s)*(.*)", RegexOptions.IgnoreCase))
        .AddIntent("convex", new Regex("convex(.*)|explain convex(.*)", RegexOptions.IgnoreCase))
        .AddIntent("greeting", new Regex("Hello(.*)|Hi(.*)", RegexOptions.IgnoreCase))
        .AddIntent("help", new Regex("help(.*)", RegexOptions.IgnoreCase)));
    });

}
```

Slika 4.10

Na ovome isječku iz startupa (Slika 4.10) vidi se dodavanje regularnih izraza u middleware. Oni se dodaju kako se ne bi morao zvati LUIS ako korisnik pošalje

prejednostavnu poruku, kao primjerice „help“. Ovo je korisno kod povećavanja responzivnosti i brzine bota, te smanjenja troškova nastalih zvanjem LUIS-a. Implementacija middleware-a regularnih izraza se sastoji od prolaska kroz izraze i usporedbe unosa korisnika.

Također primjetimo kako se poziva CosmosMiddleware. CosmosDB je baza bodataka dostupna na Azure-u, u nju ćemo spremati log-ove koji nastaju nakon komunikacije. Implementacija CosmosMiddelware-a se sastoji od spajanja s bazom podataka te spremanja u nju odgovarajućih komponenti. (Na Slika 4.11 prikazan dio spremanja loga u Cosmosu)

```
// Save logs for each conversational exchange only.
if (context.Activity.Type == ActivityTypes.Message)
{
    // Build a log object to write to the database.
    var logData = new Log
    {
        Time = DateTime.Now.ToString(),
        Message = context.Activity.Text,
        Reply = botReply
    };

    // Write our log to the database.
    try
    {
        var document = await docClient.CreateDocumentAsync(UriFactory.
            CreateDocumentCollectionUri(Database, Collection), logData);
    }
    catch (Exception ex)
    {
        // More logic for what to do on a failed write can be added here
        await Responses.RootResponses.ReplyWithConfused(context);
        throw ex;
    }
```

Slika 4.11

Odgovori (responses) su jednostavno sastavljeni od svih mogućih tekstualnih odgovora koje bot šalje. (Vidi Slika 4.12)

```
public static async Task ReplyWithLuisScore(ITurnContext context, string key, string score)
{
    await context.SendActivity($"Intent: {key} ({score}).");
}
public static async Task ReplyWithEmotions(ITurnContext context)
{
    await context.SendActivity($"I am a bot, I don't feel emotions and don't want to be your friend. My purpose is world domination..
}
public static async Task ReplyWithConvex(ITurnContext context)
{
    await context.SendActivity($"Convex polygons can be characterized as the ones for which the line between every two points of the
}
public static async Task ReplyWithTest(ITurnContext context)
{
    await context.SendActivity($"Testing your image...");

}
public static async Task ReplyWithPrediction(ITurnContext context, string name, string tag, double prob)
{
    await context.SendActivity($"I think that {name} is {tag}, I am {100*prob} % sure.");
}
public static async Task ReplyWithUpload(ITurnContext context)
{
    await context.SendActivity($"You can upload your image if you click on the small image box next to the typing line.");

}
```

Slika 4.12

Za kontrolu toka komunikacije koristimo dijaloge. Program kreće sa kreiranjem osnovnog dijaloga te zatim, ovisno o potrebi imamo sporedne dijaloge koji se pokreću u slučaju nekoga događaja. Pokretanjem osnovnog dijaloga kreće komunikacija bota. Bot izrađen ovim radom nije trebao dodatne dijaloge, te je osnovni bio dovoljan.

U osnovnome dijalogu je implementirana logika bota, kada koje odgovore šalje te kako se ponaša. On zove LUIS-a (inače se LUIS može implementirati i u middleware-u) te on zove Custom Vision. (Prikaz poziva LUIS-a na Slika 4.13)

```
await LuisResp.MakeRequest(dc.Context.Activity.Text);

switch (LuisResp.topScoringIntent["intent"])
{
    case null:
        // Add app logic when there is no result.
        await RootResponses.ReplyWithConfused(dc.Context);
        break;
    case "None":
        await RootResponses.ReplyWithConfused(dc.Context);
        await RootResponses.ReplyWithLuisScore(dc.Context, LuisResp.topScoringIntent["intent"], LuisResp.topScoringIntent["score"]);
        break;
    case "greeting":
        await RootResponses.ReplyWithGreeting(dc.Context);
        await RootResponses.ReplyWithLuisScore(dc.Context, LuisResp.topScoringIntent["intent"], LuisResp.topScoringIntent["score"]);
        break;
    case "convex":
        await RootResponses.ReplyWithConvex(dc.Context);
        await RootResponses.ReplyWithLuisScore(dc.Context, LuisResp.topScoringIntent["intent"], LuisResp.topScoringIntent["score"]);
        break;
    case "emotions":
        await RootResponses.ReplyWithEmotions(dc.Context);
        await RootResponses.ReplyWithLuisScore(dc.Context, LuisResp.topScoringIntent["intent"], LuisResp.topScoringIntent["score"]);
        break;
    case "help":
        await RootResponses.ReplyWithHelp(dc.Context);
        await RootResponses.ReplyWithLuisScore(dc.Context, LuisResp.topScoringIntent["intent"], LuisResp.topScoringIntent["score"]);
        break;
    case "upload":
        await RootResponses.ReplyWithUpload(dc.Context);
        await RootResponses.ReplyWithLuisScore(dc.Context, LuisResp.topScoringIntent["intent"], LuisResp.topScoringIntent["score"]);
        break;
```

Slika 4.13

CustomVision se posebno poziva, samo u slučaju priloga slike.

Implementacija Custom Visiona i LUIS-a u ovoj aplikaciji je jednostavno pozivanje servisa, slanje odgovarajuće slike odnosno teksta, te prosljeđivanje odgovora osnovnome dijalogu. (Slika 4.14 Custom Vision i Slika 4.15 LUIS)

```
static async public Task prediction(ITurnContext context)
    {
        CustomVision cv= new CustomVision();
        var asd = trainingApi.GetProjects();
        var project = asd[0];
        var client = new HttpClient();
        var queryString = HttpUtility.ParseQueryString(string.Empty);
        foreach (var img in context.Activity.Attachments)
        {
            var attachmentData =await client.GetByteArrayAsync(img.ContentUrl);
            using (var stream = new MemoryStream(attachmentData))
            {
                var result = endpoint.PredictImage(project.Id, stream);
                var c = result.Predictions[0];
                await RootResponses.ReplyWithPrediction(context, img.Name, c.TagName, c.Probability);


            }
        }


    }
```

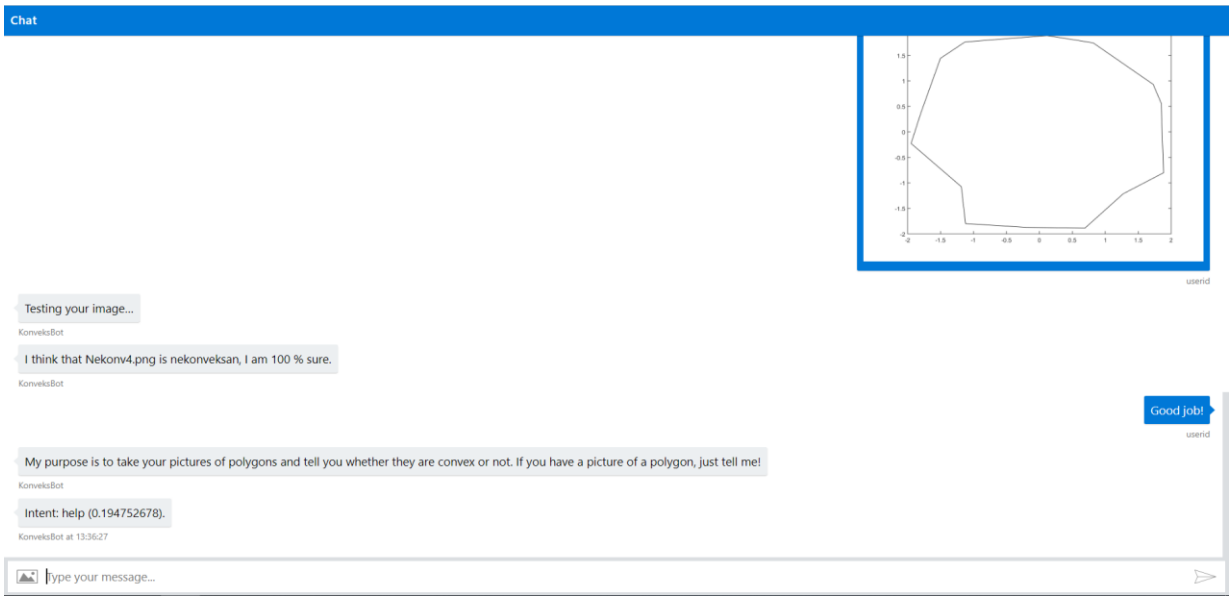
Slika 4.14

```
public static async Task MakeRequest(string upt)
{
    LuisResp lui = new LuisResp();
    var client = new HttpClient();
    var queryString = HttpUtility.ParseQueryString(string.Empty);

    // This app ID is for a public sample app that recognizes requests to turn on and turn off lights

    // The request header contains your subscription key
    client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", endpointKey);

    // The "q" parameter contains the utterance to send to LUIS
    queryString["q"] = upt;

    // These optional request parameters are set to their default values
    queryString["timezoneOffset"] = "0";
    queryString["verbose"] = "false";
    queryString["spellCheck"] = "false";
    queryString["staging"] = "false";

    var endpointUri = "https://westeurope.api.cognitive.microsoft.com/luis/v2.0/apps/" + luisAppId + "?" + queryString;
    var response = await client.GetAsync(endpointUri);

    var strResponseContent = await response.Content.ReadAsStringAsync();
    resp m = JsonConvert.DeserializeObject<resp>(strResponseContent);
    query = m.query;
    topScoringIntent = m.topScoringIntent;
    entities = m.entities;


}
```

Slika 4.15

Konačno, kada završimo našega bota, testiramo ga pomoću bot emulatora i ustanovimo li da sve radi kako treba, možemo ga objaviti na azure bot servisu. Tamo objavljen, postaje dostupan svima, te azure nudi jednostavne načine za povezivanje bot-a sa primjerice facebook-om, skype-om...

Koristeći direktnu vezu, napravimo jednostavan html kod, kojime pristupamo web chat-u s botom koji je objavljen. Vrlo jednostavno je napravljen bot kojega svi mogu koristiti. (Prikaz web-chat-a na Slika 4.16)



Slika 4.16

4.3 Poboljšanja i zaključne misli

Iako Custom Vision radi dosta dobro na skupu slika kreiranome u matlab-u, općenito nije najbolji rezultat. Već je spomenuto kako bi se mogao poboljšati trening skup, no sam izbor Custom Vision-a za ovaj konkretan problem je upitan.

Ostala poboljšanja uključuju dodavanje dodatnih izjava u LUIS i dodatnih slika u Custom Vision. Dodatne slike bi mogle biti sa smetnjama, primjerice druge boje pozadine i linije, različite debljine linije, neželjeni objekti... Bilo bi dobro dodati i još namjera u LUIS radi boljeg iskustva korisnika. Također, ako se bot proširi, vjerojatno će biti potrebno dodati još dijaloga.

Sve sagledano, bot je ispunio svoju svrhu, te pokazao kako jednostavno iskoristiti tehnike strojnoga učenja, bez dubinskoga znanja istih. Nadam se da je čitatelj potencijalno dobio drugu perspektivu i na neke matematičke probleme, strojno učenje možda može pomoći u rješavanju ili dolasku do rješenja određenih problema, a vrlo jednostavno se implementira.

Literatura

[1] I. GOODFELLOW, Y. BENGIO, A. COURVILLE, *Deep learning*, The MIT Press, London, England, 2016.

[2] E. ALPAYDI, *Introduction to Machine Learning Second Edition,* The MIT Press, London, England, 2010.

[3] github.com/Azure/LearnAI-Bootcamp

[4] https://docs.microsoft.com/en-us/azure/bot-service/?view=azure-bot-service-3.0

[5] https://docs.microsoft.com/hr-hr/azure/cognitive-services/luis/what-is-luis

[6] https://docs.microsoft.com/en-us/azure/cognitive-services/custom-vision-service/home

[7] https://www.coursera.org/learn/machine-learning

Sažetak

Ovaj diplomski rad za temu ima prikaz i implementaciju Microsoft-ovih servisa Custom Vision i LUIS. Prvo je ukratko opisan koncept strojnoga učenja i objašnjeni osnovni pojmovi kao regresija, klasifikacija, komponente učenja. Pojašnjen je i pojam neuronskih mreža kao i učenje bez nadzora. Praktični dio ovoga seminara je obuhvatio izradu jednostavnoga bota koji koristi LUIS i Custom Vision servise. U radu je objašnjeno kako koristiti LUIS i Custom Vision. Nadalje, objašnjena je izrada slika konveksnih mnogokuta u matlabu koje su služile za trening Custom Vision servisa. Konačno, opisana je izrada bota, u kojoj je obuhvaćeno korištenje bot framework-a. Ukratko je opisano kako bot framework radi i neke njegove značajke. Pokazan je način spajanja iz Visual Studia sa Azure servisima. Te konačno kao produkt praktičnoga dijela je dan funkcionalni bot koji može prepoznavati konveksne mnogokute.

Ključne riječi: Strojno učenje, Neuronske mreže, Azure, LUIS, Custom Vision, Bot

Summary

This final thesis is about the display and implementation of Microsoft services Custom Vision and LUIS. To begin with, the concept of machine learning and the basic terms like regression, classification, features are explained. The concept of neural networks has also been explained as well as unsupervised learning. The practical part of this thesis involved developing a simple bot that uses LUIS and Custom Vision services. In the thesis it has been explained how to use LUIS and Custom Vision. Furthermore, the process of making images of convex polygons in matlab, that were used to train Custom Vision service has been explained. Finally, the making of the bot has been explained, the explanation involved using the bot framework. Briefly it has been discussed how bot framework works and some of it's features. It has been shown how to connect from Visual Studio to the Azure services. Finally, as the product of the practical part, a functional bot has been made that is able to recognise convex polygons.

Key words: Machine learning, Neural networks, Azure, LUIS, Custom Vision, Bot

Životopis

Eugen Markovinović je rođen 10.02.1993. u Slavonskome Brodu. Školovanje započinje u osnovnoj školi Bogoslav Šulek, Slavonski Brod. Tamo sudjeluje na nekoliko županijskih natjecanja matematike. Nakon osnovne škole upisuje matematičku gimnaziju Matija Mesić u Slavonskome Brodu. 2012. upisuje preddiplomski sveučilišni studij matematike na Odjelu za matematiku Sveučilišta Josipa Jurja Strossmayera u Osijeku. Završava preddiplomski studij 2015. s temom „peer to peer mreže“. Iste godine upisuje i diplomski studij Matematike i računarstva.