Woodfinder

Här finns information om alla filer som ingår i projektet Woodfinder. Beskrivningarna ligger i bokstavsordning.

Översikt

Det mesta av alla menyer och annat finns i html-dokumentet, med undantag för vissa delar (t.ex. attributmodal finns i html, men den fylls dynamiskt med JavaScript). Alla knappar och liknande får sina funktioner från klasser som originerar från client på något sätt. All JavaScript börjar med index.js som skapar en instans av classen *Client.js*, och anropar samtidigt Client.initialize funktionen som sedan initierar alla andra klasser, direkt eller indirekt.

index.html

Innehåller all html som behövs för sidan. Alla modals som finns på sidan ligger här och visas endast när de behövs.

style.css

Innehåller all stil för sidan

AddLayerModal.js

Hanterar den modal som används för att lägga till layers till nätverket. Den lägger till funktionen addLayer på *add*-knappen i modalen, som tar ut inställningarna från modalen och skapar ett lager i nätverket och relevant gui. Funktionerna *modifyLayer* och *removeLayer* är funktioner som läggs till på knapparna som skapas för varje lager.

AddPolygonClassModal.js

När denna klass instansieras lägger den till *addClass*-funktionen till *add*-knappen i addClass-modalen, som tar ut klassnamnet från input rutan i modalen och lägger till som klass i nätverket och anropar sen clients *updateClassContainers*-funktion som uppdaterar alla listor och containers som ska visa alla klasser.

AttributeModal.js

Klass som hanterar attributes-modalen. Varje gång den öppnas gör den en kontroll om det finns attributer att fylla listan med. **Attributer finns inte innan man ritat ut det första träningsområdet**, då koden hämtar första pixeln från träningsområdet och hämtar ut dess attributer. Största delen av koden i klassen är till för att generera html-kod som används för

listan med alla attributer, deras checkbox-status, deras min-max toggle-status, och deras custom min-max värden. Det finns även kod som genererar ett input-fält där man kan ange custom attributer med speciell syntax.

Client.js

En client-instans skapas av index.js när sidan laddar in, och direkt efter anropas dess funktion initialize (Anledningen till varför den använder initialize istället för en konstruktor är att client behöver skickas med till en del av de klassinstanser som den skapar, och **this** går inte att använda i konstruktorn). Initialize initierar de flesta klasser som kommer att användas på sidan. De saker som inte direkt initieras av initialize initieras indirekt av den. Exempelvis initierar client en *sidebar*-instans som dels initierar sina egna knappar (t.ex. settings- och hjälpknappar) och instansierar även classer som *SidebarNetworkTab* och *SidebarPolygonsTab*, som initierar sina egna knappar.

Utöver att client instansierar en del klasser sparas även flera variabler, listor och dicts som används på flera ställen runt om i koden (mest för inställningar för nätverket som vilka attributer som ska användas, custom min-max värden för attributer osv). Client skickas med till flera olika klassinstanser då den krävs för att dessa instanser ska dela på data.

Client har även två funktioner, en funktion som försöker sätta API parametrar genom att hämta data från cookies eller en configfil, samt en funktion för uppdatering av diverse containers när man ändrar i class-listan. **Alla klasser som har en container som innehåller alla klasser har även en updateContainer funktion som anropas av den funktionen.**

Inställning av API kan göras via hemsidan under “settings”. Vid uppstart försöker Client.js att hämta de cookies som behövs för API. Om någon cookies inte kan hittas försöker client hämta data från config.json under root. Saknas samtliga kan hemsidan fortfarande användas, men utan externa anrop mot server. Felaktig config.json kommer kasta ett exception.

Configuration.js

Configuration hanterar både uppladdning och nedladdning av filer såväl som skapande och borttagande av cookies för hantering av inställningar för API calls. Eftersom det inte är möjligt att direkt ladda ned och spara filer från webbläsaren så används en blob för att skapa en fil för nedladdning som användaren själv får döpa och spara. När config filen läses in till webbläsaren så måste ordningen på attributerna vara i rätt ordning för att de ska kunna läsas korrekt och användas. Listan över attribut skapas i konstruktorn och påverkar samtliga funktioner i klassen.

CustomVariableHandler.js

Gör vad namnet antyder. Den tar en egengjord variabel för att använda till nätverket och kör det genom ett regex för att städa den inmatade informationen och göra det till en formel som går att köras i det neurala nätverket.

DataHandler.js

Datahandler annoterar den rådata som finns efter ett anrop mot databasen som används. Den kollar vilka pixlar som tagits ut och vilken klass polygonerna som finns i datamängden har och ger dem den klassen för att underlätta vid träning. Den förbereder även lösningen för ett nätverk så att alla klasser som finns utritade läggs upp i en array med nollor (0) för att användas i nätverket. Ex. Om polygon klasserna är "Skog", "Moln", "Vatten", så kommer listan att se ut enligt följande : [0, 0, 0]

DownloadPolygonsModal.js

Denna klass sätter funktion till *Save file*-knappen på downloadPolygons-modalen som använder sig av *Configuration*-klassen för att ladda hem alla polygoner med angivet filnamn.

DownloadTrainedNetworkModal.js

Klassen sätter funktion till *Save file*-knappen i *downloadTrainedNetwork*-modalen som plockar ut angivet filnamn och skickar till klassen *NeuralNetwork*s funktion *saveNetwork* som dels använder sig av tensorflows egna funktion model.save för att ladda ner en fil med en beskrivning av nätverket(antalet noder osv) och en fil som anger vikten den har lagt på lagren och dess noder. Sen skapar den även en egen fil som laddas hem mha en blob, som innehåller ytterligare information om nätverket, som min och max värden som nätverket tränades på, vilka attributer som användes, vilka klasser m.m. Alla dessa tre filer är det som krävs för att ladda upp och köra ett tränat nätverk.

Vi har haft en del problem med nerladdningen av nätverk, ibland kommer inte params-filen ner (ibland tankas den fast som en javascript void(0) objekt/fil), ibland sparar den på lite underlig ordning. Vi letar fortfarande efter orsaken till dessa problem, men vi har misstankar om att det beror på vilken webbläsare man kör, och vilka inställningar man har på nerladdning.

ElementHandler.js

Innehåller flera hjälpfunktioner skrivna med jQuery som ska underlätta att göra dynamiska ändringar i dokumentet.

Index.js

Index.js instansierar ett client-objekt och anropar dess initialize-funktion.

JstsHandler.js

JstsHandler konverterar polygoner som är ritade till Jsts-objekt för att kunna jämföra mellan polygoner och se så att de inte korsar varandra på sätt som inte tillåts.

LoadingModal.js

LoadingModal är den modal som används när träning körs för att räkna ut och visa hur lång tid det återstår innan nätverket är tränat. Det finns även en tidsindikator som kan ge en visuell representation av förloppet för träningen. Här visas även vilken förluster (loss) och med vilken precision (precision) vi har på nätverket under träningen.

MapApiHandler.js

Här hanteras anrop till den databas som används. Inställda parametrar som har gjorts i Client.js skapar ett anrop för att hämta ut den data för det område som ska undersökas. Här används adressen för anropen, en token, vilket område som efterfrågas och vilken typ av geometri som förväntas (ex wkb_geometry), samt en övre gräns för hur många objekt vi vill hantera. Funktionen getDataApi() hämtar rådata enligt den maximala yttre rektangeln för området, medan getData() sorterar ut den data som representerar polygonen. Det är rekommenderat att använda **getData()**, som också är async.

MapView.js

MapView innehåller de kart-element som syns på hemsidan. Kartan består av tre olika lager. Det första lagret representerar en karta som hämtas externt. Medan andra och tredje lagret består av de polygoner som ritas ut på kartan. Andra lagret består av ritade polygoner och tredje av det resultat som kommer ut från ett tränat AI. Det finns en hjälpfunktion som tar bort en specifik polygon från kartan. DrawInteraction() har kod som dels ser till att en polygon inte genomskär en annan polygon och gör kontroll på om autodraw är igång eller inte. Vad som händer med korrekt utritade polygoner beror på om autodraw är aktiverad eller inte. Vid aktiverad autodraw sparas polygonen direkt till bestämd klass, annars måste användaren manuellt bestämma en klass.

MapView sparar polygoner i datatypen VectorSource och skickar dem även till polygonCollection som sparar dem för GUI syften.

ModalHandler.js

ModalHandler är en superklass till alla modal-klasser som innehåller show, hide och andra funktioner som används av modals.

ModifyLayerModal.js

Den klass som används för att visa modifyLayer-modalen och populera med information om lagret. När man sen klickar save använder koden lagrets id för att ändra dess inställningar.

NeuralNetwork.js

Hanterar information tillhörande det neurala nätverket samt alla funktioner relaterat till det, som till exempel funktioner som sätter nätverkets inputdata och outputdata, kompilerar modellen, tränar och predictar.

NNElementHandler.js

Hjälpklass som skapar ett GUI-objekt för varje lager i modellen och lägger till i listan med lager.

NormalizeAttributes.js

Hjälpklass som används för att hitta min-max för attributer i datan och använder dess min-max för att normalizera värdena till något mellan 0-1.

PolygonClassHandler.js

Denna klass används för hanteringen av klasser, t.ex. när man lägger till klasser används denna klass för att skapa GUI-objekt i listan av klasser för klassen.

PolygonCollection.js

Används för att hålla koll på alla polygoner användaren ritat ut eller importerat, samt klasser och stiler för polygonerna. Finns även en hel del funktioner för att spara, ta bort eller får data om polygoner.

PolygonElementHandler.js

Denna klass hanterar att skapa GUI-objekt för polygonen i rätt klass-lista under SidebarPolygonsTab, med modify och remove knappar.

PolygonModal.js Ändra namn till AddPolygonModal.js typ

Hanterar addPolygonClass-modalen. Den får in en polygon med sin show-function och sparar den till vald klass. Den använder sig även av PolygonClassHandler för att lägga till klasser.

Popover.js

Hjälpklass som gör att man lätt kan skapa bootstrap popovers på element, för t.ex. felmeddelanden.

RandomGenerator.js

Hjälpklass som innehåller statiska metoder för diverse randomisering som används runt om i koden, t.ex. för färger och unika id.

SaveNetworkModal.js

Denna klass hanterar modalen som sparar nätverksinställningar, inte tränade nätverk utan just inställningarna.

SettingModal.js

Här sätts inställningar för anrop mot databaser. Man kan direkt ange parametrar för adress, token/API-nyckel, geometri, och en övre gräns för hur många element vi vill ta emot. Här går det även att ställa in om vi vill fortsätta att spara våra inställningar (vilka i det fallet sparas som cookies) eller att ladda upp en lokal config fil. Vid start av sidan så sker alltid ett försök att först läsa in cookies från hemsidan för att sedan läsa in en config fil från roten av program-mappen. Det går även att ladda ner de inställningar man har gjort till en ny config fil som man sedan kan ersätta sin gamla med. Sist men inte minst så finns det en “clear” knapp för att rensa cookies och förinställda parametrar.

Sidebar.js

Den klass som hanterar Sidebaren och instansierar de olika tabbarna och sätter funktion knapparna för användarmanualen och config.

SidebarHandler.js

Hanterar växlandet för sidebar, dvs visar och döljer den.

SidebarNetworksTab.js

Sätter funktion till alla knapparna som finns i tabben.

SidebarPolygonsTab.js

Sätter funktion till de knappar som finns i tabben.

Topowebb.js

Hanterar anrop mot den källa som ska användas för att få ut den karta som används för att rita polygoner på.

TrainedNetworkExtractor.js

Denna klass används för att extrahera ett tränat nätverk från uppladdade filer. Den använder dels tensorflows egna funktion tf.loadModel som tar in nätverks- och vikt-filen och sätter NNs model till det som loadModel returnerar. Dels tar den även in params-filen som den sen extraherar data (t.ex. använda klasser, min-max värden osv) om nätverket från och sätter clients listor och dicts till som sedan används vid predict av det tränade nätverket.

UploadNetworkModal.js

Används för att hantera modalen som laddar upp inställningar för nätverk och sätter NNs inställningar till det uppladdade mha NN-funktionen *setNetworkSettings*.

UploadPolygonsModal.js

Används för att hantera modalen som laddar upp polygoner. Den extraherar polygoner ur filen och tar ut dess klasser och även klassernas stil som används vid utritning på kartan.

UploadTrainedNetworkModal.js

Hanterar modalen som används för att ladda upp tränade nätverk. Den läser in filerna som laddades upp och skickas sen till NN.extractTrainedNetwork.

UserManualModal.js

Hanterar modalen som ska visa användarmodalen.