جلسه اول:

الگوریتم یادگیری میاد به فضای فرضیه نگاه میکند و یکی از آنها را که سازگار با داده های آموزشی هست انتخاب میکند. فرضیه همان فانکشن هدف ما هست که ورودی را به خروجی نگاشت میکند.

پرسپترون یک رابطه خطی مدل میکند. پرسپترون به مقادیر ورودی یک وزنی اختصاص میدهد و میگوید از این آستانه بالاتر بود فلان اگر نبود فلان. مثلا وام بدهیم یا ندهیم. آستانه عضو فضای فرضیه هست.

اگر داده های ما جدایی پذیر خطی باشند پرسپترون در تعداد متنهای میتواند این داده ها را از هم جدا کند.

در یادگیری غیر نظارتی به صورت ذاتی میتوانیم داده ها را با خطوط به خوشه ها تبدیل کنیم که تجمع داده در آنجا زیاد است.

موارد قبلی همه به صورت اپیزودیک بود و مقطعی بود یعنی این مشتری خوب هست یا نیست، اما در یادگیری تقویتی ما یک سری گام داریم و خروجی ما ترتیبی و بر اساس یک سلسله تصمیمات هستش. اینجا مسئله تصمیم گیری دنباله دار هست و تصمیمات فعلی در آینده هم تاثیر گذار است. و یک میزان مطلوبیت برای خروجی خودمان داریم و هدف یک پالیسی هستش که به ازای هر ورودی چه اکشنی را انجام بدهیم تا به مطلوبیت برسیم.

گاهی اوقات ممکن است یادگیری امکان پذیر نباشد و برای وقتی هست که ما اطلاعات کمی از تابع ناشناخته داریم و در این حالت یادگیری غیر ممکن است و برای این مواقع ما فضای فرضیه را سعی میکنیم محدود کنیم به آن مواردی که میشناسیم تا یادگیری امکان پذیر شود.

پرسپترون اون بردار وزن را میچرخاند. آستانه یک هایپر پارامتر هست. دقت کن آستانه هم عین وزن ها یاد میگیریم که همان bias ما هست که در 1 ضرب میشود و میگیم از اون مقدار کمتر یا بیشتر باشد جدا میکنیم.

چرا آمار و احتمال؟ چون از یک سری داده میخواهیم یک قاعده کلی استخراج بکنیم. و تعمیم بدهیم به کل دیتا داشتیم . اون مواردی که از روش یاد گرفتیم داده آموزشی ما هستش.

جلسه دوم:

یادگیری زمانی استفاده میشود که 1. الگو وجود داشته باشد 2. نتوانیم به صورت ریاضی بنویسیم 3. داده برای اون الگو داریم. یادگیری نظارتی میگه قرار است ما ورودی را به خروجی نگاشت کنیم که این تابع f ما هست پس ما باید از فضای فرضیه یک تابعی به نام g را انتخاب کنیم که به اندازه کافی به تابع f نزدیک باشد. تابع ناشناخته همان f ما هست. learning میگوید حتما باید فضای فرضیه را کامل بگردی نه اینکه خود فرضیه را ثابت فرض کنی و به شما بدهند.

تابع g را learning پیدا کرده است. دقت کن که بر اساس اصولی که بهش رسیدیم اگر m خیلی بزرگ باشد یعنی اون فضای فرضیه ما اون عدد 1 میشود و یادگیری امکان پذیر نمیشود چون اون مقدار باید به صفر میل بکند تا ما خطا نداشته باشیم و حدسی که میزنیم با داخل کوزه یکسان باشد پس اگر تعداد n که داده های ما هست کم باشد به همان اندازه فضای m هم باید محدود در نظر بگیریم تا یادگیری ممکن شود.

در پرسترون فضای فرضیه نامحدود است و عملا این قاعده ممکن نیست.

جلسه سوم:

هر چه قدر فضای فرضیه بزرگتر نزدیک تر کردن Ein , Eout سخت میشود. Ein میشود ورودی و برداشتی من. Eout میشود اون تخمین واقعی از کل مسئله.

چرا Ein و Eout فاصله میگیرند گاهی اوقات؟ (نمودار رنگ صورتی و سبز منظورمه) چون داده جدید میبینیم نسبت به قبلی ها در ترین و امکان misclassify شدن هست در اون خطی که پرسپترون میکشد و تا اپدیت بکند در این نتیجه داده ها به خوبی جدایی پذیر خطی نیستند.

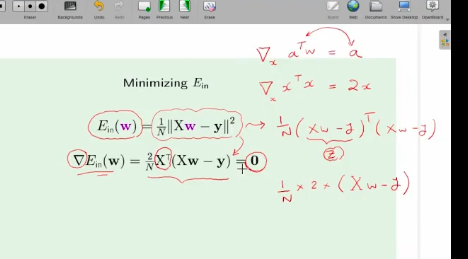
چرا پرسپترون خوب جدا نکرده؟ چون ممکن است در جای بدی متوقف شده باشد یا early stopping کرده باشد و تو جای بدی متوقف شده باشد و اگر چند تا iteration بیشتر میزد خطا کمتر بود.

تغییر در الگوریتم پرسپترون: بخاطر وجود این مشکل میایم یک تغییری ایجاد میکنیم، و میگیم کدوم یک از مدل ها کمترین Ein داشته و اون رو به عنوان جواب در نظر میگیریم. که به این الگوریتم میگوییم pocket که یعنی اون مدل بهتر را در جیب خودمان گذاشتیم و جلو میرویم و در نهایت از اون بهترین استفاده میکنیم.

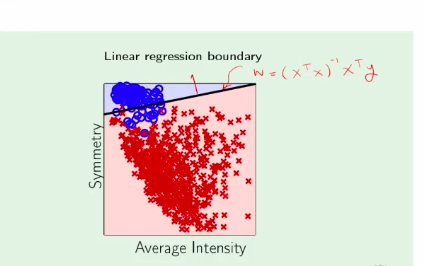
Regression: خروجی یک کلاس نیست یک مقدار حقیقی است. مثلا مقدار وامی که قرار است به آن شخص بدهیم. اون توان 2 علامت را خنثی میکند حالا چه h(x) بزرگتر از f(x) باشد یا نباشد. در این دست مسائل اگر Ein را کوچک بدست آوردی تقریبا میتوانی مطمئن شوی که Eout هم کوچک است. در دسته بند های خطی به جز وقتی که x خیلی بزرگ باشد بعد خیلی بزرگ باشد. ولی اگر کوچکتر از N باشد جای نگرانی نیست.

اگر ضرب داخلی اون بردار در وزن مثبت شد میشود جز کلاس 1 و اگر نشد میشود کلاس -1. X مثلا چون زاویه آن کمتر از 90 درجه هست میشود مال -1. اگر بیشتر از 90 درجه باشد 1. بردار وزن متعامد هست به زیر فضای خطی اون زیر فضای خطی ابر صفحه هست و مقدار ضرب داخلی آنها صفر میشود. تنها زیر فضای خطی که بعدش یکی کمتر از فضا هست میشود ابر صفحه.

نرم 2 به توان 2 تک به تک درایه ها را به توان 2 میرساند و با هم جمع میزد. که میشود همان 2 تا خط اطراف همون وزن منهای خروجی واقعی. برای کم کردن هم مشتق میگیریم.



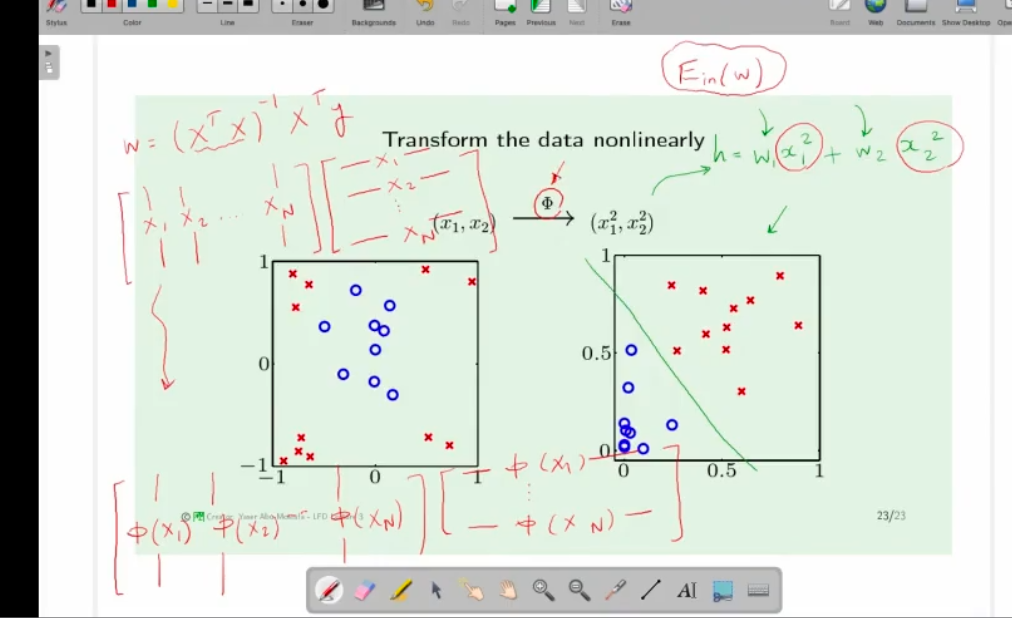
رنک ماتریس میشود تعداد ستون هایی که از همدیگر مستقل خطی هستند. برای وقتی هست که تعداد داده ها خیلی بزرگتر از بعد هست و فیچر ها باید مستقل از هم باشند. فیچر ها نباید به همدیگه وابستگی خطی داشته باشند اگر این را تعریف کنیم میتوانیم بگوییم رنک ماتریس حداکثر d+1 شود که به معنی full rank بودن هست پس معکوس پذیر هست.



از regression برای کلاس بندی استفاده کردیم اما اشکال کجاست؟ چون در پایین قرمز ها مثلا خروجی تابع وزن در ورودی میشود -3 ولی این تابع میخواست -1 اتلاق کند به این نقاط و MSE زیاد میشود این باعث میشود دچار خطا شود و اذیت شود برای همین به یک جواب مطلوب نمیرسیم و معمولا از این برای وزن اولیه یا جواب اولیه برای الگوریتم پرسپترون استفاده میکنند.

مسائل در خطی با SINGLE SHOT حل میشوند یعنی کلا ورودی ضربدر وزن میکنی و دیگر iteration ندارد. مثلا برای مقدار ماندن در خانه طرف بعد 5 سال فرار نمیکند ولی این مقدارش با 20 سال در حالت خطی فرقی نمیکند یعنی اهمیتی ندارد 5 سال تا 20 سال ولی هر چه قدر بیشتر باشد این یعنی ماندگاری بالاتر است و برای ما مهم است پس میایم از مسائل غیر خطی استفاده میکنیم و مثلا میگیم طرف آیا کمتر از 1 سال و بیشتر از 5 سال ساکن بوده است؟

وقتی یک regression خطی هست نسبت به چی هست؟ نسبت به وزن ها یا ورودی؟ نسبت به وزن ها خطی هست دیگه چون داریم نسبت به وزن مشتق میگیریم. برای کلاس بندی هم همین است. حالا ورودی را میخواهیم ثابت فرض میکنیم و یک transformation غیر خطی بزنیم روی داده ها (اینم مشکلی نداره چون خطی بودن نسبت به وزن ها هست نه نسبت به ورودی ها مثلا فاصله داده از مراکز) و به فضای جدیدی برویم که و درفضای جدیدی جدای پذیر خطی هستند.



جلسه چهارم:

در regression خطی با یک گام میتوانیم مسائل را حل بکنیم که میشود وزن برابر با pseudo inverse ضربدر y که همان ورودی ما هستش. خطی بودن نسبت به وزن ها هست و نه نسبت به ورودی ها و مزیتی ندارد پس برای غیر خطی کردن تابع دسته بند یا classifier این است که بر روی ورودی یک تابع غیر خطی اعمال کنیم به فضای جدید برویم و اون ورودی را وارد کنیم و میتوانیم مسائل و الگو های پیچیده تر راه حل بکنیم نسبت به حالت صرفا خطی.

تمام مشتق گیری نسبت به وزن ها هستند.

پس چیشد در فرمولی که برای آلفا ضربدر بردار ستونی که میزان شباهت ورودی با تک به تک داده ها هست چیکار میکند؟ ضرب داخلی میدهد و مشخص میکند و حاصل میاد ورودی را با تک به تک آموزشی شباهت را میسنجد و به هر شباهت یک وزنی اختصاص میدهد و میاد وزن ها را جمع میزند تا score اون داده را بدست بیاورد و مشخص کند به کدام کلاس تعلق دارد اگر مثبت بود یک کلاس اگر منفی بود کلاس دیگر.

البته میزان تاثیری داریم یعنی اگر شباهت بیشتر بود ضریب بیشتری باشد و تاثیر اون برچسب زیاد تر باشد تا جمع مثبت شود و برود به کلاس مثبت ها و اگر منفی بود ضریب کمتر و میرود یک کلاس دیگر و کلاس بند انجام میشود. الفا برچسب داده i ام خواهد بود. این ضرب از regression شروع شد بعد الفا بدست اومد.

ضرب ها رو دقت کن اگر بردار سمت چپ ماتریس بود روی سطر کار میکند اگر سمت راست ماتریس بود دارد روی ستون کار میکند و تاثیر دارد. این y که در فرمول آلفا هست میخواهد بگوید که برای اینکه برچسب نقطه j را پیدا بکنی بیا تمام نقاط آموزشی را در نظر بگیر و ببین شباهت اون نقطه به نقطه j که همان ضرب داخلی هست چه قدر هست مقدار را ضربدر آلفا بکن پس اگر داده قرار هست مثبت باشد برای کلاس مثبت باشد آلفا آن باید مثبت باشد یعنی به داده هایی شبیه هست که آلفا های آنها مثبت هست و اگر نقاط مشابه را الفا آنها را جمع بزنی به برچسب اون نقطه برسی. یعنی اگه برچسب به مثبت ها نزدیک بود یعنی مثبت هست دیگه.

حالا اگر y منفی بود بیا داده هایی که منفی هستند و آلفا آنها منفی هست را جمع بزن و k های آن باید بزرگ باشند دیگه.

وقتی داده ورودی را غیر خطی میکنی به یک نمایش جدید از ورودی میرسی. ماتریس k درایه های آن ضرب داخلی داده ها هستند. و ضرب داخلی نمایش جدید میشود kernel که ماتریس کرنل هم گفته میشود. از فی برای غیر خطی کردن بدست میاد و حاصل فقط به ضرب داخلی وابسته میشود نه خود فی و ضرب داخلی آنها مهم است واسه همین میشود کرنل که یک بازنمایی جدید از همین regression ها خواهد بود.

دقت کن در یادگیری ماشین دنبال یک تابع فرضی هستیم که دقیقا اون تخمین مپ کردن ورودی به خروجی را انجام دهد حالا خطا چیست؟

Error: دنبال خطای فرضیه ما هستیم. و اکثر اوقات از یک تعریف point wise استفاده میکنیم و میایم نگاه را به جای اینکه به کل داده ها نگاه بکنیم به یک داده نگاه میکنیم و بر اساس اون میبینیم چه قدر فرضیه ما خطا دارد مثلا اگر مسئله regression باشد از square error استفاده میکنیم اگر classification باشد اگر عبارت داخل براکت صحیح بود 1 میشود یعنی برابر بود و گرنه صفر و دسته بندی انجام میشود.

میانگین خطا هم میایم متوسط گیری میکنیم و خطا ما روی داده های آموزشی ما هست. حالا اگر روی جمعیت کل داده ها به جای آموزشی استفاده بکنیم e out را بیرون خواهد داد. Pointwise کجا باید باشد؟ در قسمت ارزیابی فرضیه نهایی ما خواهد بود.

معیار error: معیار خطا بر اساس مسائل مختلف، مختلف باید باشد.

Friendly measure: اگر جزئیات ریز ندانیم و میایم صرفا برای راه حل ساده تر استفاده میکردیم مثلا از square loss برای کلاس بندی استفاده میکردیم و اگر استفاده میکردیم در یک گام میتوانست w0 خوب به ما بدهد. مثلا در مسئله دسته بندی خطی اگر از entropy cross استفاده بکنیم بهینه سازی را ساده تر میکند و بهینه سازی محدب میشود و این بهینه سازی به شکل عددی پیدا کردن اون واقعا کار راحتی هست یا در حالت بسته برای regression مناسب است.

پس برای دسته بندی از cross entropy برای تابع loss استفاده میکنیم.

Error measure هم برای ارزیابی سیستم مهم است هم در قسمت یادگیری تا به یک فرضیه خاص برسیم در یادگیری و بعد اون فرضیه را ارزیابی کنیم.

Noise: گاهی اوقات ممکن است که target function تابع نباشد، یا الگو تابع نباشد. مثلا پیشبینی بر اساس آب و هوا مثلا برای ساعت 10 شب ممکن است آب و هوا یا ابری یا بارونی باشد که این در تضاد با مفهوم تابع است که میگفت به ازای یک ورودی مشخص یک مقدار مشخص برگردانده میشود. پس میایم به صورت احتمالی نگاه میکنیم به تارگت فانکشن که تابع نیست و میایم توزیع نگاه میکنی یعنی یک مشتری 90 درصد خوب هست و 10 درصد بد هست بعد میتوانیم تصمیم گیری بکنیم. به این توزیع تارگت distribution گفته میشود. در نهایت noise میشود اون انحرافی که از مقدار متوسط داریم. هر چه قدر عدد بالا تر نویز بیشتر هست چه منفی چه مثبت. اگر تابع بود ارتباط خروجی با x میتوانیم بگیم در همه مقادیر صفر است به جز جایی که برابر با y است. الان هدف ما این است که توزیع تارگت را یاد بگیریم که p y | x را یاد بگیریم که target distribution ما خواهد بود و تا الان نیازی نداریم در یادگیری نظارت شده بیایم و p x را یاد بگیریم که توزیع داده های ورودی ما هستند.

کلا هدف چی هست برای امکان پذیر بودن یادگیری؟ Ein خطای اون فرضیه ما به ازای Eout آن یکسان باشد به ازای همان فرضیه به این generalization bound گفته میشود تا فرضیه مطمئن باشیم روی داده های جدید هم درست کار بکند. در واقع به نوعی دنبال این هستیم که خطا صفر شود خطای چی ؟ Eout به ازای اون فرضیه ای که ما استفاده کردیم. اندازه فضای فرضیه ما مهم هست و اگر پیچیده باشد امکان پذیر نخواهد بود یادگیری. اون کرنل برای این بود که خطا صفر شود و اون generalization برای این بود که خطای فرضیه نمونه ما به ازای کل یکی شود. بعضی ها هم از هر دو روش به صورت توام استفاده میکنند. در پرسپترون بی نهایت وزن ممکن داریم و تعداد هایپر پلین ها به صورت بی نهایت هست در صورتی که قرار بود فضای فرضیه ما محدود باشد. VC dimension: پیچیدگی فضای فرضیه را مدل میکند برای وقتی که فضای فرضیه به سمت بینهایت میرود و حالت صعودی دارد و در ادامه Ein کم و Eout به سمت زیاد شدن حرکت میکند. Generalization gap مقدار بین Ein و نمودار Eout خواهد بود. هم Ein باید کم شود هم فاصله تعمیم پذیری تا Eout مناسبی داشته باشیم. اگر پیچیدگی فرضیه کم باشد اصلا نمیتواند مدل کند پس Ein زیاد هست هر چه قدر پیچیده تر میشود این مقدار کمتر میشود و بهتر مدل میکند از یه جایی به بعد اینقدر با سرعت پیچیده میشود که خطای out بیشتر میشود. گفتیم پیچیدگی بالا برود خطا کم میشود دیگه حالا فرض کن فی ما اینقدر بتواند بعد بالا ببرد و پیچیدگی ایجاد بکند تا بینهایت که خطای ما صفر شود ولی از طرف دیگر ممکن است overfit شود و generalization bound ما خراب شود. kij یعنی شباهت بین iوj چون ضرب داخلی آنها هست.

**جلسه پنجم:**

Point wise میاد و برای مقدار Ein در یک نقطه مشخص یعنی ورودی x خطا را میسنجد که میزان واقعی اختلاف خروجی واقعی نسبت به فرضیه ما هست. و اگر این میانگین را روی کل داده های جمعیت خودمان بگیریم امید ریاضی نسبت به x را Eout یا خطای کلی نسبت به تمام جمعیت بدست خواهد آمد.

Training and testing:

دقت کن حالا هدف این هست که برای تعداد بینهایت عضو بررسی میکند اون boundیی که قرار هست بدست بیاوریم.

مشکل چی بود؟ مشکل این بود در اون linear bound مقدار همپوشانی بسیار زیاد هست و حالتی نخواهد بود که این مجموعه های bad event جدا از هم دیگر باشند به همین دلیل اگر m ما به سمت بینهایت برود که تعداد فرضیه های ما هست خروجی بسیار بد خواهد بود و bound ما دارد با تعداد فرضیه ها تغییر میکند و linear bound راهکار خیلی شدیدی را در پیش میگیرد.

Ein اون خطای ما نسبت به داده ها خواهند بود که اشتباه مرز بندی شده اند و Eout میشود اون ناحیه ای که مرز ما اشتباه تصمیم گیری کرده و شده 2 تا مثلث چون اون ناحیه ای است که دارد خطا رخ میدهد Ein خطای sample ما هست و میخواهیم بدونیم اون داده هایی که اشتباه کلاس بندی شده اند. M سایز فضای فرضیه ما خواهد بود.

Dichotomy میاد تابع فرضیه را روی تعداد نقاط محدودی از ورودی اعمال میکند به جای اینکه مثل فرضیه روی تمام ورودی اعمال شود.

تابع رشد: بیا حداکثر dichotomy های رو n تا نقطه پیدا بکن یعنی این n تا نقطه را جوری انتخاب بکن که بیشترین pattern را از h استخراج بکنی یعنی ببین این n تا نقطه کدوم بیشترین dichotomy رو ایجاد میکند. یک انتخاب بد انتخاب خطی بودن هست چون اصلا باعث میشود همه مقدار یکسانی بگیرند و اشتباه هست و حداکثر الگو بدست نمیاد.

مجموعه محدب: اگر هر 2 تا نقطه را داخل آن در نظر بگیریم و خط بکشیم تمام اون خط در داخل اون هست. برای اینکه به مقدار رشد خیلی بالا مثل 2 به توان n برسیم لازم نیست که حتما فضای فرضیه کل توابع ممکن باشد فقط کافی است فضای فرضیه به اندازه کافی پیچیده باشد و شکل های پیچیده تولید کند به این نوع حالت با یک مجموعه محدب میگوییم داریم به کرانه بالا میرسیم به ازای n تا نقطه. غیر محدب ها هم سوپر ست محدب ها هستند پس چون بیشتر از 2 به توان n محدب ها نمیشوند برای غیر محدب هم در نهایت به این مقدار میرسیم. شمارش تمام pattern ها از یک محدب بسیار سخت خواهد بود بنابراین سعی میکنند به جای اینکه تمام فضای فرضیه را محاسبه بکنیم میایم یک ویژگی از فضای فرضیه انتزاع میکنیم تا کرانه بالا را بدست بیاوریم نه مقدار دقیق را پس میایم فقط ثابت میکنیم m چند جمله ای هست همینکه این را ثابت کنیم یادگیری ممکن خواهد بود. Shattered نشود یعنی چی؟ یعنی به ازای اون نقاطی که داریم بر اساس فرضیه که مد نظر داریم مثلا پرسپترون نتوانیم همه الگو ها را توسط فرضیه تولید بکنیم که بهش break point هم میگوییم اون نقطه ای که دیگر بیشتر از آن نمیتوانیم همه الگو ها را با این فرضیه تولید کنیم. دونستن break point کافی است چون میدانیم یادگیری ممکن است مقدارش ضروری نیست. در حالت مسائل محدب تضمین نمیتوانیم بکنیم یادگیری را چون به سمت بینهایت میرود و breakpoint لزوما شاید وجود نداشته باشد. اگر بدانیم breakpoint باشد اثبات میکند تابع رشد به صورت چند جمله ای خواهد بود. دقت کن ما کران بالا را نمیتوانیم رد بکنیم اگر رد کنیم یعنی کران بالا درست نبوده است.

**جلسه ششم:**

M تعداد فضای فرضیه ها هست میخواستیم یک m را جایگزین بکنیم و برای این راه به این رسیدیم که bad event ها همپوشانی زیادی دارند که به dichotomies رسیدیم که میگفت که میخواهیم فرضیه ای که داریم بررسی میکنیم نتایج را روی زیر مجموعه ای از نقاط ببینیم و برای ما مهم نیست فضای حالت را چطوری دسته بندی میکند حالا چرا میتواند همپوشانی بین بد ها را پوشش بدهد؟ چون با اونهایی که همپوشانی دارد، خروجی یکسانی را دارند تولید میکنند پس میتوانیم M را جایگزین بکنیم و به یک مفهوم تابع رشد رسیدیم که فضای فرضیه به n تا نقطه محدود میشد بفهمیم چه سایزی دارد یعنی بر اساس این نقاط چند تا فرضیه میتوانیم بدست بیاوریم و میگفتیم نقاط را جوری بدست بیار که توابع فرضیه بیشینه شود و محاسبه این تابع کار دشواری هست چون به ازای n نقاط باید بدست میومد پس به break point رسیدیم که میگفتیم تعدادی از داده ها هست که در اونجا اگر به اون نقاط برسیم تابع رشد بیشینه مقدار خودش را ندارد. اگر break point که ما یک عدد محدود باشد تابع m ما چند خطی میشد و اون رشد، رشد تابع exponential را در حاصل ضرب آن به سمت صفر میبرد و خوب میشد. دقت کن 4 تا حالت نمیتواند باشد چون با پرسترون نمیتوانیم جدایی پذیر خطی کنیم واسه همین بود مقدار 4 ام را نمیپذیرفتند.

در واقع به دنبال این هستیم که اگر یک H جدید یا classifier جدید اومد باید به جای اینکه تابع رشد را حساب بکنیم با دانستن break point میتوانیم کران بالا پاسخ را بدست بیاوریم. دقت کن چند جمله ای را دقیقا پیدا نمیکنیم یک کرانی بدست میاوریم که اون کران یک چند جمله ای خاص هست و این تابع رشد جایگزین M میشود.

Break point اگر از order n باشد یعنی اصلا break point نداریم چون ممکن است تا بینهایت زیاد شود.

چون به ازای هر فرضیه ناحیه bad event های آن با هم همپوشانی دارند پس همپوشانی ها را در نظر گرفته و خیلی بهتر از union bound میشوند و فضای اشغال شده آن کمتر میشود. union bound اشتراک نمیگیرد و اجتماع bad event ها را میگیرد واسه همین کل فضا را میگیرد . اگر فرضیه ها خروجی یکسانی یا الگوی یکسانی تولید بکنند برای داده ها bad event ها یکسان میشود اگر از Eout صرف نظر بکنیم فعلا.

جلسه هفتم:

Vc dimension میشود بیشترین مقدار N یی که تابع رشد بیشترین مقدار خودش را بگیرد. یعنی تا این مقدار میتواند تمام الگو ها را بسازد یا فرضیه وجود دارد که آن را تولید بکند. Bound ما مستقل از یادگیری هست و مستقل از ورودی هست چون به صورت دلخواه نقاط را انتخاب کردیم و فرض خاصی نداشتیم و با هر توزیعی اون bound برقرار هست. تابع هدف ما هم هر چیزی باشند هم این bound برقرار است. فضای فرضیه و فرضیه نهایی و مجموعه آموزشی را نمیتوانیم حذف کنیم در تئوری امروزی این قسمت learning هم وارد خواهد شد که خروجی g هست و ویژگی یادگیری بر روی boundیی که بدست میاد تاثیرگذار خواهد بود. Vc پرسپترون کمتر مساوی d+1 است. دترمینان ماتریس مثلثی که صفر دارد جز قطر اصلی 1 است و چون دترمینان غیر صفر هست معکوس پذیر هست.

جلسه هشتم:

Shatter کردن یعنی بتواند همه را درست برچسب دهی بکند. درجه آزادی پارامتر هایی هستند که ما میتوانیم تغییر بدهیم تا توابع مختلف بسازیم. پارامتر های ما در پرسپترون در درجه آزادی آنالوگ هستند چون اعداد حقیقی دارند اما VC میگوید که شما چند تا پارامتر آزاد دارید جا اینکه چند تا توابع میتوانید بسازید. همیشه تعداد پارامتر های آنالوگ با تعداد درجه آزادی باینری همیشه یکی نیست. مثل پرسپترون که درجه آزادی بیشتر میشود اما کمکی نکرده است توابع مختلفی بسازیم پس VC dimension مفهوم ضعیف تری هست نسبت به درجه آزادی.

Generalization bound:

Sample complexity: چه قدر sample نیاز هست که learning در یک setupیی اتفاق بیفتد.

Bias variance trade-off:

Eout ما متغیر تصادفی هستش چون بر اساس دیتا ست ما وابسته هست و با هر دیتا ست فرق میکند پس میایم متوسط ازش میگیریم تا یک عدد به ما نمایش دهد به جای متغیر تصادفی.

Expected value صرفا داری انتگرال میگیری از متغیر تصادفی و جابجایی انتگرال ها فرمول حسابان دارد که میتوانیم ترتیب انتگرال ها را تغییر بدهیم یکی از آن ویژگی ها کران دار مثبت هست که میتوانیم جابجایی انتگرال را انجام بدهیم.

جلسه 24:

جلسه 28:

k-means در فرمول محاسبه فاصله یکبار بهینه سازی را s را ثابت و نسبت به میو بهینه سازی را انجام میداد و بار دیگر برعکس این کار را میکرد.

وقتی داریم نسبت به c بهینه سازی میکنیم باید مشخص کنیم هر داده به کدام خوشه نزدیک تر است که در واقع مثلا با 3 تا فضا به 3 ناحیه تقسیم میشود حالا نسبت به مراکز دسته ها یا خوشه بهینه سازی ها انجام میشود. بعد دوباره نسبت به c بعد دوباره نسبت به مراکز خوشه ها تا ثابت شود خوشه ها و داده ها. نسبت به مراکز اولیه خوشه ها حساس است و دلیلی ندارد نسبت به کمینه سراسری بهینه شود پس بهتر است از چندین نقطه استفاده بکنیم.

بعد از مشخص شدن خوشه یک داده میایم میانگین میگیریم از داده های آن خوشه و مرکز جدید را انتخاب میکنیم.

در اول کار نمیدانیم هر داده به کدام خوشه تعلق دارد پس چون نمیدانیم اسم را expectation میگذاریم و اسم مرحله بعد را maximization میگذاریم چون داریم میانگین میگیریم که پاسخ بهینه سازی کمینه فاصله داده ها تا مرکز هست.

فازی همان soft assignment است.

مشکل چی بود؟ ما فرض میکردیم x یک توزیع نرمال دارد که با یک میو و سیگما نشان بدهیم و تتا پارامتر های این توزیع بودند و با MLE پارامتر ها را پیدا میکردیم. مشکلی که پیش میومد این بود که توزیع داده ها گوسی نباشند و مشکل داشتیم یک راه این بود که بریم سراغ روش های بدون پارامتر. یک راه دیگر این بود که ما فکر کنیم توزیع داده ها ترکیبی از گوسی های مختلف است. هر توزیع پایه ای هم میتواند باشد برای سادگی توزیع نرمال را به عنوان توزیع پایه ای در نظر گرفته ایم.

تعداد پارامتر ها به تعداد داده ها مربوط است پس روش بدون پارامتر است.

به این روش که از جمع یک سری توزیع گوسی استفاده کردیم به عنوان mixture modeling گفته میشود. یکی از مشکلات این روش این است که k را نمیتوانند scale کنند با داده ها و باید داده ها را ثابت در نظر بگیریم و بعد k پیدا کنیم.

پای ها یا همان عدد پی نسبت ترکیب از توزیع های مختلف هستش در همان جمع توزیع های گوسی. برای نمونه برداری از این توزیع اول یک توزیع از پی بردار که یک توزیع احتمال روی اعداد 1 تا k میدهد بعد یکی از نمونه ها را بردار و برو از mixture iام یک نمونه از این توزیع برداریم که میتواند به هر کدام از این خوشه ها به اندازه احتمالات تعلق داشته باشد که همان soft assignment است.

Latent variable modeling: از روی n تا داده میخواهیم خوشه بندی کنیم و یک اطلاعات نهانی است که نمیتوانیم ببینیم این است که وقتی میخواستیم داده تولید شود از کدام خوشه استفاده شده است به نام y و ما نمیتوانیم آن را ببینیم چون در داده ها ندیدیم. پای ها احتمال پیشین هستند که داده i به کلاس y تعلق داشته باشد. Y=i همان متغیر نهان ما است که مثلا میگوید به خوشه i تعلق داشته.

پارامتر ها را نمیدونستی MLE بزن.

اگر مسئله ای که گفته شد را به جای soft assignment بیایم و hard assignment در نظر بگیریم به همان مسئله k-means میرسیم و دیگر پای i ها مقدار یا صفر یا یک دارند و مقادیر احتمالاتی ندارند. چه راه حلی برای بیشینه کردنش است؟ اولی این است که نسبت به ناشناخته مشتق بگیریم که پیچیده میشود و غیر خطی. یک روش دیگر این است که از کاهش گرادیان استفاده بکنیم تا پارامتر ها اپدیت بشود که کند است و راهکار سوم این است که از EM استفاده بکنیم.

EM: یک روش iterative است مثل گرادیان کاهشی یک فرقی که دارد این است که در مرحله e میایم اون متغیر نهان را استنتاج میکنیم به ازای هر داده بر اساس تخمین فعلی از پارامتر و با استفاده از این استنتاج داده ها را به صورت soft assignment به خوشه های مختلف نسبت میدهیم و MLE میزنیم. این همچنان این مشکل را دارد که بهینگی محلی همگرا شود ولی سریعتر است. دقت کن میزان تعلق به خوشه ها از قبل مشخص است فقط مراکز خوشه ها مشخص نیست. دقت کن اینجا likelihood کامل هست به ازای همه داده ها مثل قبلی نیست که y را نهان میکرد. و این log تصادفی هست بخاطر برچسب چون برچسب ها را ما نمیدانیم و چون میخواهیم تصادفی بودن را حذف بکنیم، امید ریاضی گرفتیم نسبت به اون برچسب ها که تصادفی هستند. q log likelihood هست روی داده های کامل و یک امید ریاضی زدیم روی اون بخشی که نداشتیم.

پیدا کردن نزدیک ترین خوشه برای یک داده با استفاده از تخمین فعلی روی پارامتر های ناشناخته صورت میگیرد و بعد از آن یک جمع وزن دار داریم برای پیدا کردن مراکز.

جلسه 28:

یادگیری تقویتی:

دنبال این هستیم که action هایی که میتوانیم در محیط انجام بدهیم در یک محیط دینامیک را بهینه بکنیم بر اساس بازخوردی که از محیط داریم. در model-free control مدل را نمیشناسیم.

RL چیست؟ خیلی از اوقات تصمیم فعلی ما در آینده تاثیر گذار است و اثر local ندارد بلکه اثر بلند مدت دارد. رویکرد های قبلی همگی تصمیم گیری موضعی داشتند مثلا کلاس این داده چی باید باشد. هدف تقویتی این است که یاد بگیریم اکشن ها را چطوری انجام بدهیم یا به شکل دقیق تر شرایطی که توش هستیم در دنیای واقعی را نگاشت بکند به اعمال و این کار را به گونه ای انجام بدهد که پاداشی که از محیط دریافت میشود بیشینه شود یا مجموع این پاداش بیشینه شود. environment محیطی است که agent توش اکشن انجام میدهد و محیط بر اساس پاداش از وضعیت سیگنال به ما آگاهی میدهد.

یک فرضی راجب محیط داریم که میشود با MDP تقریب بزنیم. MDP میگوید ما بر اساس دنباله ای از شرایط تصمیم گیری میکنیم آخرین وضعیت در این دنباله ی ما تمام اطلاعات کافی برای تصمیم گیری در آینده را دارد.

در فرآیند مارکوف یک متغیر داریم اما در MDP ما 2 متغیر داریم یکی state و دیگری action ها هستند. که چون میخواهیم اکشن انجام بدهیم پس ما تصمیم گیری باید بکنیم. اکشن درست اکشنی است که پاداش ما را بیشینه بکند.

اکشن روی آینده ما تاثیر گذار است و ما باید نتایج اکشن ها را پیشبینی بکنیم.

با استفاده از مفهوم reward به عامل میگیم خوب عمل کرد یا نکرد.

Sparse reward: اگر کار بدی کرد بهش پاداش منفی میدیم و گرنه چیزی بهش نمیدهیم و اگر کار خوب انجام داد بهش پاداش میدهیم.

Policy: نگاشتی از حالات محیط به اکشن ها است. به گونه ای که تجمیع شده reward ها را میخواهد بیشینه بکند.

یعنی ما یک راهنمایی به سیستم دادیم و سیستم کور نیست ولی کامل نظارت داریم صرفا اعمال را نظارت میکنیم.

اکشن عامل را انجام میدهد بعد محیط وضعیت جدید و reward را به عامل میدهد. p اطلاعات state و reward را میگیرد و اکشن را انجام میدهیم که به این policy گفته میشود.

MDP:

P یک توزیع احتمالی است و transition به صورت احتمالی بر اساس اکشن و وضعیت میگیم به این احتمال میرویم به این state. یعنی کاملا قطعی نیست و شرایط کاملا یکسان نیست و مثلا شاید نتوانیم همه شرایط محیطی را درک بکنیم مثلا جاده لغزان باشد واسه همین یک شانس هم در نظر گرفتیم واسه همین حالت بعدی ما تصادفی هست.

Discount factor: از این برای تجمیع reward ها استفاده میشود. نسبت به policy میخواهیم reward ها را بیشینه کنیم. اگر reward های ما محدود باشد فرمول بیشینه کردن الزاما محدود نیست و ممکن است همگرا نشود، میشود از یک مفهوم discounting استفاده کرد. اگر گاما کوچک تر از 1 باشد و reward های ما محدود باشد یا کران داشته باشد میتوانیم با سری هندسی نشان بدهیم جمله با احتمال 1 محدود خواهد شد و همگرا خواهد شد. چون reward ها را در یک وزن کوچک تر از 1 ضرب میکنیم به این معنی هست که reward هایی که اخیر اتفاق افتاده اند مطلوبیت بیشتری دارند. چون ترجیح میدهیم اتفاق خوب زودتر اتفاق بیفتد.

اگر زمان محدود باشد discount factor را 1 میگذاریم البته به شرط اینکه اگر زودتر یا دیرتر بخواهیم بازی را تمام کنیم برای ما مهم نباشد.

Stochastic policy: بر اساس state فعلی یک احتمالی بین همه اکشن ها و میدهد و اکشن بعدی را با یک احتمالی انتخاب میکند.

در حالت ساده deterministic است یعنی بر اساس وضعیت اکشن انجام میدهیم اما در RL احتمال میدهیم به هر اکشن.

P رفتار عامل را تنظیم میکند چون هدف P این است که reward را بیشینه بکند.

Value function: Gt همان utility است که جمع discount reward ها است. در واقع میگیم بر اساس وضعیت فعلی با دنبال کردن policy p به چه مقدار utility به صورت متوسط میرسی که میشود value.

Action-value: اینجا هم دنبال متوسط utility هستیم اما یک اکشن مشخصی انجام دادیم که بر اساس policy p نبوده است چه مقدار به طور متوسط utility بدست میاری.

Bellman Equation: مرتبط میکند V را به V وضعیت های بعدی. با این دستگاه ارزش همه حالت ها را بر اساس policy p به ما میدهد بعد بر اساس value هایی که یاد گرفتیم یک فعالیت policy improvement انجام میدهیم. در نهایت خروجی باعث میشود که V فعلی به بقیه V ها مرتبط شود و چون این معادله خطی است یک دستگاه خطی ایجاد میشود چون این رابطه به ازای همه state ها برقرار است.

هدف این است که V p بیشینه شود که هدف MDP است یا به شکل معادل p را پیدا بکند که منتهی به V بیشینه شود در محیطی که از این policy تبعیت میکند.

یک مشکلی وجود دارد این است که یک policy بهینه شاید وجود نداشته باشد چون ممکن است واسه هر state یک policy باشد به policy بهینه گفته میشود که V به ازای همه stateها با یک policy مشخص بیشتر شود از V همان state به ازای هر policy دیگری بزرگتر مساوی شود. چطوری؟

میگوید actionیی را پیدا بکن که مجموع rewardیی که میگیری و بعد با رفتن به هر stateیی که رفتی بهینه رفتار بکن با هر policy که خواستی تا expected بیشینه شود. اینجا بحث ما اون p است که میخواهیم بگیم بهتر از هر policy دیگر است به ازای هر حالت مشخص. چطوری میگیم؟ چون داریم max میگیریم از اکشن اگر یک اکشن دیگر را انتخاب کنیم هم اون دیگر max نیست اون اکشنی خواهد بود که از policy دیگری آمده است. هم V\* دیگر نمیگیریم زیرا V P را داریم استفاده میکنیم. پس سعی داریم دستگاه معادله غیر خطی را حل کنیم. چرا غیر خطی است؟ چون max داریم.

چرا دستگاه هست؟ چون ارزش بهینه هر state را دارد نسبت به ارزش بهینه state های دیگر دارد مرتب میکند. این دستگاه غیر خطی را با value iteration حل میکنیم که بهش policy iteration گفته میشود.

پس به صورت کلی دنبال P\* هستیم که نسبت به همه policy ها در همه حالت ها بهتر است.

Policy iteration: از تکنیک برنامه نویسی پویا کمک میگیرد با کمک general MDP. برای اینکار دو کار باید بکنیم اول یک policy evaluation داریم که یک policy غیر بهینه را انتخاب میکنیم و ارزیابی میکنیم بعد سعی میکنیم یک ذره بهتر کنیم بعد دوباره برمیگردیم ارزیابی میکنیم نسخه بهتر شده را.

مرحله اول policy evalutation معادل با حل کردن یک دستگاه خطی هست چرا خطی چون دنبال بهینه نیستیم دنبال یک حالت غیر بهینه هستیم.

مرحله دوم policy improvement هست که از یک مفهوم greedy کمک میگیرد. دنبال این هستیم که اکشنی را انتخاب بکنیم که expected reward بر اساس policy p میگیرد را بیشینه بکند یا به عبارت دیگر اگر در این حالت این اکشن را انجام بدهی بعد از آن policy p را دنبال بکنی بیشترین reward را بهت بدهد. این باعث میشود ارزش تمام state ها یا بزرگتر شود یا تغییر نکند. چطوری؟ اول به ازای یک state خاص میاد تغییر ایجاد میکند و تغییر به این شکل است که این p ارزیابی شده را روی A میاد بیشینه میکند اون هم روی یک state خاص فقط و با بقیه کاری ندارد، چه تغییری در توابع ارزش ایجاد میشود؟ بیشتر میشود چرا؟ چون جمله اول reward فقط زیاد میشود و V آن دارد بیشتر میشود جمله دوم اگر به state هایی دیگر رفتیم تغییر نمیکند یا اگر اون حالات منتهی بشوند به حالت فعلی که توش بودیم ارزش اونها هم بالا میرود چون ارزش فعلی بالاتر شده است پس در مجموع در نهایت ارزش همه state ها بیشتر میشود که بهش policy improvement گفته میشود و این کار را بر اساس پالیسی که ارزیابی کردیم انجام دادیم. و حالا برمیگردیم پالیسی پریم را ارزیابی میکنیم و همینجوری این کار را ادامه میدهیم تا به یک پالیسی بهینه همگرا میشود. چرا؟ چون زمانی هست که پالسی بهینه تغییر نکند و اگر پالیسی تغییر نکند باید حتما در معادله bellman صدق بکند که به این معنی هست که به V\* رسیدیم و پالیسی بهینه رسیدیم. حالا چرا همگرا میشود؟ چون دنباله افزایشی داریم که اگر کران دار باشد حتما همگرا میشود.

یک راه دیگر این است که به نام greedy improvement که میگوید بیا پالیسی را تصادفی فرض کن و بیا اون اکشنی که q را بشینه میکند در نظر بگیر تا به اکشن های دیگه یک شانس انتخاب شدن بدهیم و باعث میشود ارزش همه state ها را افزایش میدهد. چرا؟ یک شانسی به بقیه action ها میدهد تا بتواند Q بهینه را پیدا بکند.

Value iteration: به جای قبلی که سعی میکند یک پالیسی غیر بهینه پیدا کند و اون رو ارتقا دهد به صورت مستقیم سعی میکند دستگاه غیر خطی را حل کند که این دستگاه میشود value for state میشود بیشینه روی اکشن های مختلف از expected value تابع utility که میشود reward فعلی بعلاوه گاما در reward های آتی ما هست. برای حل این دستگاه غیر خطی باید از fixed point iteration استفاده بکنیم که میگوید سمت راست را حساب کن بعد به v بده همین کار را iterative انجام بده تا به جواب برسیم یعنی دو تا منحنی با هم تلاقی پیدا میکنند. بدی این روش این است که هزینه اینها زیاد است روش قبلی یک دستگاه خطی حل میکردیم و order کمتری داشت.

Contraction mapping را سرچ کن.

Prediction ارزش حالت ها هستش. Control یعنی policy بهینه را پیدا میکند و به جواب بهینه میرسیم.

همه این 3 روش از مفهوم پویا استفاده میکنند چون ما پاسخ های stateها را تا یک لحظه ای داریم بعد به وسیله اینها میتوانیم ارزش جدید state را پیدا کنیم با جایگذاری در معادلات بازگشتی. و این ارزش ها را در یک جدولی ذخیره شده اند بخاطر همین میگیم برنامه نویسی پویا.

همه چیزایی که تا الان خوندیم نیاز داشت MDP را کامل بشناسیم هم transition probability ها و هم reward ها را بدانیم اما در دنیای واقعی اینها را نمیدانیم مثلا نمیدانیم رقیب ما چطوری واکنش نشان میدهد یا واکنش محیط چی است. پس باید با محیط تعامل بکنیم پس بحث exploration پیش میاد و ما یک سری اکشن هایی که بهینه نباشند را انجام میدهیم تا نسبت به محیط شناخت پیدا کنیم و یک تخمینی از تابع MDP بزنیم. که بهش model-free prediction گفته میشود.

Model-free prediction: بدون دانستن اون 2 تا بیایم و V P (S) را یاد بگیریم. ساده ترین این کار است که در هر episode یعنی ( از یک حالت اولیه شروع میشود یک دنباله ای از اکشن ها اجرا میشود و در یک جایی قطعش میکنیم) بررسی بکنیم اولین مواجه ما با حالت s است و در اون اولین مواجه یک متغیر به نام N S را مقدار دهی بکنیم که تعداد بار های ملاقات ما با اون حالت هست و هر بار که ملاقات کردیم این مقدار را اپدیت بکنیم تحت policy p.

چطوری v s را اپدیت کنیم ؟ در v s قبلی ضربدر تعداد مشاهدات قبلی بعلاوه g t الان دیدیم جمع بکنیم بعد تقسیم بر تعداد مشاهدات بکنم که همان میانگین گیری است و به شکل iterative اپدیت میشود. هر وقت ما به حالت s رسیدیم باید تا آخر episode دنباله اکشن ها را بریم تا g t بدست بیاد که مجموعه reward ها تا لحظه s است پس باید تا آخر episode انجام بدهیم. میانگین گیری به expected value همگرا میشود یا میانگین گیری معمولی به میانگین واقعی همگرا میشود و نه از transition و نه از reward استفاده کردیم و فقط از تجربیات استفاده کردیم که g t است. این روش MC نام دارد جز روش های model-free prediction است.

جلسه 29: