سوال ۱.

الف

$$y_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \quad \text{for} \quad j = 1 \text{ to } n$$

در صورتی که یکی از z_i ها برابر بی‌نهایت شود، softmax به ۱ میل می‌کند چون تمام مقادیر غیر از z_i به ۰ میل می‌کنند.

ب

مدل پرسپترون یکی از مدل‌های پایه و مبنایی شبکه‌های عصبی مصنوعی است و به عنوان یک نمایش ساده از نورون‌های بیولوژیکی در نظر گرفته می‌شود. این مدل برای یادگیری نظارت‌شده کلاس‌بندی باینری (تقسیم داده‌ها به دو کلاس) استفاده می‌شود. پرسپترون یک بردار از ویژگی‌های ورودی را دریافت می‌کند و به هر ویژگی وزنی اختصاص می‌دهد. سپس یک جمع وزن‌دار از ورودی‌ها محاسبه می‌شود و تابع فعال‌سازی را بر روی آن اعمال کرده تا خروجی تولید شود.

$$output = activation\ function(w_1 \times x_1 + w_2 \times x_2 + ... + w_n \times x_n + bias)$$

حالا که دو ورودی داریم پس برای خروجی داریم:

$$output = activation\ function(w_1 \times x_1 + w_2 \times x_2 + bias)$$

بله، این کار امکان‌پذیر است. پرسپترون می‌تواند توابع منطقی ساده مثل AND و OR را تقلید کند. در زیر، این کار را برای هر دو تابع با وزن‌ها و آستانه بیان می‌کنیم.

AND Perceptron

بله، این کار امکان‌پذیر است. پرسپترون می‌تواند توابع منطقی ساده مثل AND و OR را تقلید کند. در زیر، این کار را برای هر دو تابع با وزن‌ها và آستانه بیان می‌کنیم. AND Perceptron برای یک پرسپترون که تابع منطقی VÀ را تقلید کند، ما می‌توانیم وزن‌ها و آستانه (بایاس) را به شکل زیر تعیین کنیم:

$$w_1 = 1, \quad w_2 = 1, \quad \text{bias} = -1.5$$

در این صورت، برای ورودی‌های (۰, ۰)، (۰, ۱)، (۱, ۰) خروجی ۰ و برای ورودی (۱, ۱) خروجی ۱ خواهیم داشت، که دقیقا عملکرد گیت AND است.

OR Perceptron

برای یک پرسپترون که تابع منطقی OR را تقلید کند، ما می‌توانیم وزن‌ها و آستانه را به شکل زیر تعیین کنیم:

$$w_1 = 1, \quad w_2 = 1, \quad \text{bias} = -0.5$$

در این صورت، برای ورودی $(0, 0)$ خروجی ۰ و برای ورودی‌های $(0, 1)$، $(1, 0)$، $(1, 1)$ خروجی ۱ خواهیم داشت، که دقیقا عملکرد گیت OR است.

در عمل، مدل پرسپترون توسط الگوریتمی مثل gradient descent آموزش داده می‌شود تا وزن‌ها و بایاس را پیدا کند. ولی در این مثال ساده، ما می‌توانیم به صورت دستی آن‌ها را مشخص کنیم.

XOR Perceptron

یک پرسپترون تنها نمی‌تواند تابع XOR را مدل کند. دلیل این است که XOR یک تابع که غیرخطی است و پرسپترون‌ها فقط قادر به مدل‌سازی توابع خطی هستند. در واقع، این یکی از محدودیت‌های اصلی پرسپترون‌ها است که منجر به توسعه شبکه‌های عصبی چندلایه شده است.

با این حال، می‌توانیم یک شبکه‌ی پرسپترون چند لایه (MLP) با استفاده از دو گیت AND و OR و یک گیت NOT بسازیم که تابع XOR را مدل کند.

مدل MLP به این صورت خواهد بود:

لایه‌ی اول (پنهان): دو نورون، یکی برای گیت AND و دیگری برای گیت OR :

$$AND\ Perceptron : w_1 = 1, w_2 = 1, bias = -1.5$$

$$OR\ Perceptron : w_1 = 1, w_2 = 1, bias = -0.5$$

لایه‌ی دوم (خروجی): یک نورون با گیت NOT برای خروجی AND و OR از لایه‌ی پنهان:

$$NOT\ Perceptron : w_1 = -2, w_2 = 1, bias = 0.5$$

در اینجا، خروجی گیت AND در ورودی گیت NOT قرار می‌گیرد. به این ترتیب، گیت NOT خروجی گیت AND را انکار می‌کند، در حالی که گیت OR بدون تغییر باقی می‌ماند. این منجر به تابع XOR می‌شود.

سوال ۲. بله، الگوریتم Naive Bayes Classifier یکی از الگوریتم‌های مهم و پرکاربرد در یادگیری ماشین است. در این الگوریتم، احتمال اینکه یک نمونه به یک دسته خاص از دسته‌های مسئله تعلق داشته باشد، بر اساس احتمالات شرطی و احتمالات آپریوری محاسبه می‌شود.

فرض کنید C مجموعه‌ای از k دسته مسئله باشد و $X = (x_1, x_2, \ldots, x_n)$ برداری از n ویژگی مربوط به یک نمونه باشد. هدف Naive Bayes Classifier ، یافتن دسته‌ای است که بهترین احتمال اینکه نمونه با ویژگی‌های X به آن تعلق داشته باشد، را دارد.

برای این کار، ابتدا احتمالات آپریوری $P(C)$ برای هر یک از دسته‌ها محاسبه می‌شود. سپس برای هر یک از ویژگی‌ها، احتمال شرطی $P(x_i|C)$ برای هر دسته و هر ویژگی محاسبه می‌شود. به عبارت دیگر، ما به دنبال پیدا کردن احتمال این هستیم که ویژگی x_i در دسته C وجود داشته باشد. این احتمالات با استفاده از داده‌های آموزشی محاسبه می‌شوند.

با توجه به این احتمالات، ما می‌توانیم احتمال اینکه نمونه با ویژگی‌های X به دسته C تعلق داشته باشد، را برای هر دسته C محاسبه کنیم. این احتمال برای هر C به شکل زیر است:

$$P(C|X) = \frac{P(C)\prod_{i=۱}^{n} P(x_i|C)}{P(X)}$$

در اینجا، $P(C|X)$ احتمال شرطی برای دسته C به شرط داشتن ویژگی‌های X است، $P(C)$ احتمال آپریوری دسته C است، $P(x_i|C)$ احتمال شرطی برای ویژگی x_i در دسته C است و $P(X)$ احتمال ظاهر شدن ویژگی‌های X در تمام دسته‌ها است.

احتمال آپریوری $P(C)$ و احتمال شرطی $P(x_i|C)$ با استفاده از الگوریتم‌های مختلفی محاسبه می‌شوند، از جمله الگوریتم‌های تخمین تک بعدی Gaussian ، تخمین تک بعدی Bernoulli و تخمین چند بعدی Gaussian . در ضمن، فرض نادرستی بسیار ساده‌ای که در این الگوریتم اعمال می‌شود، فرض نادرستی Naive Bayes نام دارد. این فرض به این صورت است که تمام ویژگی‌ها مستقل از یکدیگر هستند، به عبارت دیگر، هیچ تداخلی بین آن‌ها وجود ندارد.

در کل، الگوریتم Naive Bayes Classifier به دلیل سرعت بالا و دقت خوب در بسیاری از مسائل، از الگوریتم‌های محبوب در یادگیری ماشین است. البته، این الگوریتم در مواردی که فرض نادرستی Naive Bayes قابل قبول نیست، به خوبی عمل نمی‌کند.

الف

$$P(Spam|w_۱, w_۲, w_۳, w_۴, w_۵) = P(Spam)\prod_{i=۱}^{۵} P(w_i|Spam) = \frac{۱}{۵۴}$$

$$P(Not-Spam|w_۱, w_۲, w_۳, w_۴, w_۵) = P(Not-Spam)\prod_{i=۱}^{۵} P(w_i|Not-Spam) = \frac{۹}{۱۲۸۰}$$

در نتیجه احتمالا اسپم است.

ب

$$P(Spam|w_۱, w_۲, \neg w_۳, \neg w_۴, \neg w_۵) = P(Spam)\prod_{i=۱}^{۲} P(w_i|Spam)\prod_{i=۳}^{۵} P(\neg w_i|Spam) = ۰$$

$$P(Not-Spam|w_۱, w_۲, \neg w_۳, \neg w_۴, \neg w_۵) =$$

$$P(Not-Spam)\prod_{i=۱}^{۲} P(w_i|Not-Spam)\prod_{i=۳}^{۵} P(\neg w_i|Not-Spam) = \frac{۱}{۱۲۸۰}$$

در داده‌های داده شده هیچ نمونه اسپمی نداریم ولی طبق احتمال‌هاِ، ایمیل اسپم است.

ج

$$P(Spam|w_۱, w_۲, \neg w_۳, \neg w_۴, \neg w_۵) = P(Spam)\prod_{i=۱}^{۲} P(w_i|Spam)\prod_{i=۳}^{۵} P(\neg w_i|Spam) =$$

$$\frac{۶+۵}{۱۰+۱۰} \times \frac{۵+۱}{۶+۵} \times \frac{۴+۱}{۶+۵} \times \frac{۵+۱}{۶+۵} \times \frac{۴+۱}{۶+۵} \times \frac{۰+۱}{۶+۵} \approx ۰/۰۰۳$$

$$P(Not-Spam|w_۱, w_۲, \neg w_۳, \neg w_۴, \neg w_۵)$$

$$= P(Not-Spam)\prod_{i=۱}^{۲} P(w_i|Not-Spam)\prod_{i=۳}^{۵} P(\neg w_i|Not-Spam) =$$

$$\frac{۴+۵}{۱۰+۱۰} \times \frac{۱+۱}{۴+۵} \times \frac{۱+۱}{۴+۵} \times \frac{۱+۱}{۴+۵} \times \frac{۱+۱}{۴+۵} \times \frac{۲+۱}{۴+۵} \approx ۰/۰۰۰۳$$

پس ایمیل اسپم شناخته می‌شود.

سوال ۳.

الف

$$\frac{\partial cost}{\partial w_۰} = ۲(y - h\Theta(x))(x-۱) \times \frac{\partial h}{\partial z} \times \frac{\partial z}{\partial w_۰} = -۲(y-h)h(۱-h)$$

$$\frac{\partial cost}{\partial w_۱} = -۲x_۱(y-h)(۱-h)h$$

$$\frac{\partial cost}{\partial w_۲} = -۲x_۲(y-h)(۱-h)h$$

تابع هزینه و گرادیان های مربوطه را در نظر بگیرید، برای بروزرسانی وزن ها در هر دور تکرار از گرادیان نزولی، ما می توانیم از قاعده زیر استفاده کنیم:

$$w_{i,\text{new}} = w_{i,\text{old}} - \alpha\frac{\partial \text{cost}}{\partial w_i}$$

که در آن α نرخ یادگیری است.

اگر ما α را برابر با ۱ در نظر بگیریم، ما داریم:

$$w_{۰,\text{new}} = w_{۰,\text{old}} + ۲(y-h)h(۱-h) \quad (۱)$$

$$w_{۱,\text{new}} = w_{۱,\text{old}} + ۲x_{۱}(y-h)(۱-h)h \quad (۲)$$

$$w_{۲,\text{new}} = w_{۲,\text{old}} + ۲x_{۲}(y-h)(۱-h)h \quad (۳)$$

به طور کلی، نرخ یادگیری α در فرمول‌های بروزرسانی را تنظیم می کنیم تا کنترل کنیم که گام‌های بروزرسانی چقدر بزرگ باشند. اگر α خیلی بزرگ باشد، ما ممکن است از حداقل محلی عبور کنیم و اگر خیلی کوچک باشد، فرآیند آموزش ممکن است خیلی کند باشد.

ب

ما معمولا تابع سیگموید را به عنوان تابع فعال‌سازی در لجستیک رگرسیون یا شبکه های عصبی استفاده می‌کنیم، اما از آن به عنوان تابع هزینه استفاده نمی‌کنیم. دلیل اصلی این است که هنگامی که تابع سیگموید را به عنوان تابع هزینه استفاده می‌کنیم، ممکن است با مشکل «هزینه محلی مینیمم» (local minimal) مواجه شویم.

به طور خاص، گرادیان های تابع سیگموید ممکن است خیلی کوچک شوند، یک مشکل به نام vanishing gradients که می‌تواند منجر به توقف یا تاخیر زیادی در یادگیری شود. در حقیقت، تابع سیگموید می‌تواند برای مقادیر خیلی بزرگ یا کوچک، گرادیان های خیلی کوچکی داشته باشد که می‌تواند فرآیند یادگیری را کند کند.

به همین دلیل، ما معمولا از تابع هزینه cross-entropy در لجستیک رگرسیون استفاده می‌کنیم، که توانایی بهتری در مدیریت این مشکلات دارد. تابع هزینه cross-entropy به خوبی با تابع فعال‌سازی سیگموید ترکیب می‌شود و منجر به گرادیان های بهتری در فرآیند یادگیری می‌شود.

در این مثال و بخش ب، تابع دوم در لوکال مینیمم‌ها زیاد گیر می‌کند اگر از gradient descent استفاده کنیم چون یک تابع محدب نیست ولی اولی این مشکل را ندارد.

سوال ۴.

الف

در مدل‌سازی آماری و یادگیری ماشین، overfitting و underfitting دو مشکل مهم هستند.

Overfitting اتفاق می‌افتد وقتی مدلی داده‌های آموزشی را خیلی خوب یاد می‌گیرد تا جایی که دقت آن بر روی داده‌های جدید یا تست کاهش می‌یابد. به عبارت دیگر، مدل تمامی جزئیات و سر و صداهای موجود در داده‌های آموزشی را یاد می‌گیرد که ممکن است در داده‌های جدید یا تست موجود نباشد. این باعث می‌شود مدل قادر به تعمیم داده‌های جدید نباشد.

Underfitting اتفاق می‌افتد وقتی مدل نمی‌تواند الگوهای موجود در داده‌ها را بخوبی بیاموزد. این اغلب به دلیل سادگی بیش از حد مدل (به عنوان مثال، یک مدل خطی برای داده‌های غیرخطی) یا نبود داده‌های کافی برای یادگیری الگوهای پیچیده اتفاق می‌افتد. Underfitting باعث می‌شود مدل نتواند به خوبی بر روی هم داده‌های آموزش و هم داده‌های تست عمل کند.

برای مثال، فرض کنید که شما می‌خواهید یک مدل یادگیری ماشین را برای پیش‌بینی قیمت خانه بر اساس متراژ آن آموزش دهید. اگر مدل شما overfitting کند، ممکن است از یک مدل پیچیده بیش از حد استفاده کند که هر نوسان

کوچک در قیمت خانه‌های آموزشی را یاد می‌گیرد، اما بر روی داده‌های تست خوب عمل نمی‌کند. اگر مدل شما underfitting کند، ممکن است از یک مدل خیلی ساده استفاده کند (مثلا یک خط ساده) که نمی‌تواند تغییرات قیمت خانه را بر اساس متراژ بخوبی بیاموزد.

مفهوم overfitting در سمت راست نمودار و جایی که فاصله‌ی error ها افزایش می‌یابد، قابل مشاهده است.

مفهوم underfitting در سمت چپ بالای نمودار قابل مشاهده است.

ب

زمانی که پیچیدگی مدل کم است، خطای Validation Set و Train Set هر دو زیاد است و اصلا از نقاط درست نمی‌گذرند.

زمانی که پیچیدگی مدل بیشتر می‌شود، overfitting رخ می‌دهد و خطای Train Set از خطای Validation Set کمتر است و در نتیجه نمودار دوم شبیه‌ترین نمودار به خطای مدل بر روی داده‌های تست است.

سوال ۵.

افزایش داده‌ی آموزش: این کار می‌تواند مشکل را حل کند، به خصوص اگر مشکل overfitting باشد. با داشتن داده‌های آموزش بیشتر، مدل می‌تواند الگوهای عمومی تری را یاد بگیرد و این می‌تواند به جلوگیری از overfitting کمک کند. اما باید توجه داشت که این روش می‌تواند هزینه‌بر و زمان‌بر باشد و همچنین نمی‌تواند همیشه مشکل underfitting را حل کند.

کاهش پارامتر regularization : این کار می‌تواند به حل مشکل underfitting کمک کند. Regularization یک تکنیک است که برای جلوگیری از overfitting استفاده می‌شود، به طوری که با افزایش آن، مدل می‌تواند ساده‌تر شود. اما اگر مدل در حال حاضر underfitting دارد، کاهش regularization می‌تواند به مدل اجازه دهد تا پیچیدگی بیشتری را یاد بگیرد.

افزایش پارامتر regularization : این کار می‌تواند به حل مشکل overfitting کمک کند.

با افزایش regularization ، مدل می‌تواند ساده‌تر شود و بنابراین احتمال overfitting کاهش می‌یابد. اما اگر مدل در حال حاضر underfitting دارد، افزایش regularization می‌تواند مشکل را بدتر کند.

استخراج ویژگی‌های بهتر از داده: این کار می‌تواند در هر دو حالت، یعنی overfitting و underfitting ، مفید باشد. استخراج ویژگی‌های بهتر می‌تواند به مدل کمک کند تا الگوهای مهم‌تر و عمومی‌تری را در داده‌ها بیابد. این کار می‌تواند به جلوگیری از overfitting کمک کند و همچنین می‌تواند به مدل کمک کند تا با کمک ویژگی‌های مهم‌تر، پیچیدگی‌های موجود در داده‌ها را بهتر یاد بگیرد و بنابراین به حل مشکل underfitting کمک کند.

موفق باشید.