

NUMPY

Sıfırdan Array Oluşturma:

`np.zeros(10, dtype=int) → array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])`

`np.ones((3,5)) → array([[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1]])`

`np.full((3,5), 3) → 3'lardan oluşan 3x5lik array`

`np.arange(0, 37, 3) → 0'dan 37'e kadar 3'er 3'er artan dizisi.`

`np.linspace(0, 1, 10) → 0'dan 1'e kadar eşit aralıklı 10 sayı.`

`np.random.normal(10, 4, (3,4)) → ortalaması 10, standart sapması 4 olan 3x4lik array.`

`np.random.randint(0, 10, (3,3)) → 0-10 arası 3x3lik bir array.`

`np.ndim → boyut sayısını verir. (2 boyutlu 3 boyutlu vs.)`

`np.shape → boyut bilgisini verir. ((3,5), (4,2) gibi)`

`np.size → toplam eleman sayısını verir.`

`np.dtype → integerin tipini verir (örn: int64)`

`np.reshape → elde olan diziyi istenilen şekilde sokar (tek boyutlu diziyi 3x3 array yapmak gibi)`

`np.concatenate([x,y]) → x ve y arrayini birleştirir (daha fazla elemandan birleştirilir)`

`a = np.array([[1, 2, 3],
[4, 5, 6]])`

`np.concatenate([a,a]) → array([[1, 2, 3],
[4, 5, 6],
[1, 2, 3],
[4, 5, 6]])` → axis=0
satır bazlı, birleştirme

`np.concatenate([a,a], axis=1)`

→ sütun bazlı, birleştirme

`x = np.array([1, 2, 3, 99, 99, 3, 2, 1])`

`np.split(x, [3,5]) → [array([1, 2, 3]), array([99, 99]), array([3, 2, 1])]`

`a, b, c = np.split(x, [3,5])`

`a = array([1, 2, 3]), b = array([99, 99]), c = array([3, 2, 1])`

\bar{x} = aritmetik ortalaması

Veri Bilimi ve Machine Learning

Ölçüt Türleri:

1-) Medyan = ortanca terimin bulunması $\frac{n+1}{2}$, $\frac{(\frac{n}{2})\text{.terim} + (\frac{n}{2}+1)\text{.terim}}{2}$

2-) Mod = Bir seride en çok tekrar eden değere Mod adı verilir.

3-) Kartiller = medyanın sol ve sağ tarafındaki ortanca değerlerdir (çeyrekler)

4-) Range (Degrin Aralığı): Bir seride maksimum değer ile minimum değerin farkıdır.

5-) Standart sapma: Ortalamadan olan sapmaların genel bir ölçüsüdür.

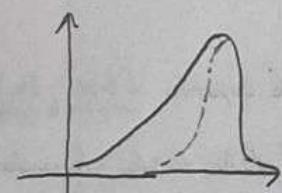
$$s = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

6-) Varyans: standart sapmanın karesidir

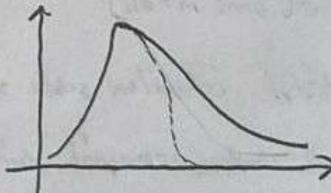
7-) Sarılıklık: Pearson sarılıklık katsayısi

$$\frac{3(\bar{x} - \text{medyan})}{\text{standart sapma}}$$

$p < 0 \rightarrow$ negatif sarılık (soldan)
 $p > 0 \rightarrow$ pozitif sarılık (sağdan)
 $p = 0 \rightarrow$ simetrik

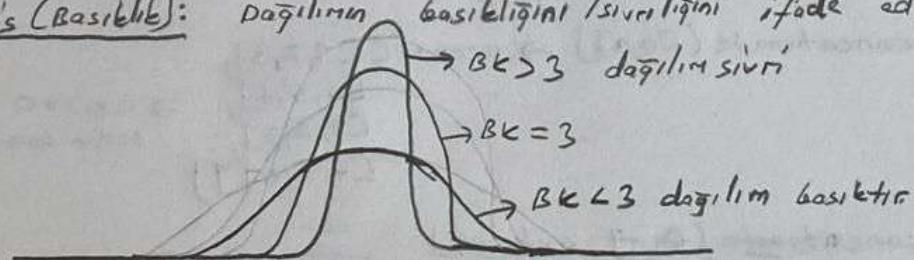


negatif sarılık



pozitif sarılık

8-) Kurtosis (Basıklık): Dağılımlının basıklığını / sivrilliğini ifade eder.



$$\frac{\sum_{i=1}^n (x_i - \bar{x})^4}{n}$$

$$\text{Basıklık katsayısi} = \frac{m_4}{s^4} \rightarrow \text{standart sapmanın 4.kuvveti}$$

NUMPY

iki boyutlu ayırma:

$m = np.arange(16).reshape(4,4)$

$np.vsplit(m, [2]) \rightarrow$ ilk 2 arrayi ve son 2 arrayi ayırtır.

sıralama:

$np.sort(x) \rightarrow x$ arrayini sıralar.

$m = np.random.normal(20, 5, (3, 3))$

$np.sort(m, axis=1)$
veya 0 (yatay ve dikey sıralama)

$np.subtract(x, 1) \rightarrow x$ arrayindeki her elemandan 1 çıkar.

$np.add(x, 1) \rightarrow x$ arrayindeki her elemene 1 ekler.

$np.multiply(x, 4)$, $np.divide(x, 2) \rightarrow$ bölme ve çarpma işlemleri.

$np.power(x, 3) \rightarrow x'in 3. kuvvetidir.$

$np.mod(x, 2) \rightarrow x \% 2$ (kalansı verir)

$np.absolute(np.array([-3])) \rightarrow$ mutlak değer işlemini yapar.

$np.sin(360)$, $np.cos(180) \rightarrow$ trigonometrik işlemler yapılabilir.

$v = np.array([1, 2, 3])$
 \downarrow toplanır toplanır
 $np.log(v)$, $np.log^2(v)$, $np.log10(v) \rightarrow$ logaritma işlemleri yapılabilir.

Numpy ile iki bilinmeyenli denklem çözümü:

$$5x_0 + x_1 = 12, \quad x_0 + 3x_1 = 10$$

$a = np.array([[5, 1], [1, 3]])$

$b = np.array([12, 10])$

$x = np.linalg.solve(a, b) \rightarrow$ denklemi kendisi çözter ve $x \rightarrow x_0$ ve x_1 sırasını verir.

PANDAS

Pandas Serisi Oluşturma:

Pandas oluşturduğu yapılarda index bilgisini de tutar.

seri = pd.Series([10, 88, 3, 4, 5])

0	10
1	88
2	3
3	4
4	5

seri.axes → index bilgisini verir

seri.head(4) → ilk 4 tane veri

seri.tail(2) → sondan 2 tane veri

pd.Series([99, 22, 332, 96, 5], index=[3, 3, 5, 7, 9])

varya
index = ['a', 'a', 'c', 'd', 'e']

pd.Series({'reg': 10, 'log': 11, 'cart': 12})

pd.concat([seri, seri]) → iki seriyi birlestirme

Pandas Dataframe oluşturma ve işlevleri:

df = pd.DataFrame(m, columns=[değisen isimleri...])
herhangi bir array

df.head() → tablodan alınan veri sayısı (ilk sıradaan başlar)

df.columns → columns bilgisini verir

df.columns = ('...', ..., ...) → yeni column isimleri tanımlanabilir.

df.drop('index', axis=0) → index satırını tamamen tablodan siler (geçici olarak)

df.drop('index', axis=0, inplace=True) → index satırını kalıcı olarak siler

df['yeni_index'] = df['varolan_index'] / df['varolan_index']

→ yeni bir satır oluşturur ve sahip olduğunuz indexler arasında işlemi sonucunu atar.

Seaborn modülü kullanılabılır.

Pandas loc ve iloc kismi:

`m = np.random.randint(1,30, size=(10,3))`

`df = pd.DataFrame(m, columns=["var1", "var2", "var3"])`

loc: tanımladığı şekli ile seçim yapmak için kullanılır

`df.loc[0:3]` → 0 ve 3 dahil olan tablonun kısmını verir

iloc: alışık olduğumuz indeksleme mantığıyla seçim yapar

`df.iloc[0:3]` → 0 dahil ama 3 dahil değil (liste mantığı)

`df.loc[0:3, "var3"]` → var3 sütunu 0 ve 3'ün dahil olduğu satırı var3'ü yapar
iloc 1'sin hata verir `df.iloc[0:3, "var3"] = error`

iloc 1'sin doğru kullanım `df.iloc[0:3][["var3"]]` dir

`df.var1` → "var1" sütununu verir

`df[df.var1 > 15][["var1"]]` → 15'den büyük satırları verir

Birleştirme işlevleri:

`df1 = pd.DataFrame({ "calisanlar": ["Ali", "Veli", "Ayse", "Fatma"],
"grup": ["Muhasebe", "Muhendislik", "Muhendislik", "IK"] })`

`df2 = pd.DataFrame({ "calisanlar": ["Ayse", "Ali", "Veli", "Fatma"],
"ilk_giris": [2010, 2009, 2014, 2019] })`

`pd.merge(df1, df2)` → bu 'ki' yapıyı ortak özelliklerine göre birleştirir

özelliklere göre tabloyu ürpürtelabilir veya kırıtabilir

`df.describe()` → count, mean, std, max, min ... gibi verileri set tabloda verir

`df.groupby("sütun ismi").mean()` → aynı veri isimlerini işlemeye tabii tutar

*** `df.apply`, `df.filter`, `df.transform` önebilir fonksiyonlar (özellikle lambda ile)

`df.select_dtypes(include = ["object"])` → object tipindeki verileri seçen

1

Supervised Machine Learning: Regression and Classification

Machine learning, yapay zekanın bir alt dalı olarak kabul edilmiştir ve gelişmeye devam etmektedir.

Bir algoritma ne kadar çok öğrenme, deneme fırsatı vererek ne kadar iyi performans gösterir?

Supervised Learning (Gözetimli öğrenme) bir çok gerçek dünya uygulamasında kullanılan ve en hızlı gelişmelerin, inovasyonların görüldüğü Machine Learning alanıdır. Supervised Learning temelinde giriş ile çıkış arasındaki ilişkiyi öğrenen algoritmadir. Gözetimli öğrenmenin ayırt edici özelliğinin bu algoritmalarla öğrenmesi için örnekler verilir. Bu örnekler verilen x değerleri için elde edilen y sonuçlarını içerir. x değerleri ve ona karşılık gelen y değerine göre algoritma, hangi x değeri için hangi y değerinin olması gerektiğini öğrenir. Bu train süreci sonucunda verilen x değerleri için y değerini tahmin eder.

<u>Input (x)</u>	<u>Output (y)</u>	<u>Application</u>
email	spam? (0/1)	spam filtering
audio	text transcripts	speech recognition
English	Spanish	machine translation
ad, user info	click? (0/1)	online advertising
image, radar info	position of other cars	self-driving car
image of phone	defect? (0/1)	visual inspection

Regression (housing price prediction, for example) predict a number infinitely many possible outputs. (sonsuz olasılıklı sayılar)

Classification (for example, breast cancer detection, whether it is malignant or benign)

Bu yapıya classification denmesinin sebebi sadece belirli sayıdaki kategoriler arasından tahmin de bulunmasızdır. (Kötü huylu tümör mü, iyi huylu tümör mü, o ya da 1 gibi)

Classificationda aynı variyeli çeşitli türlerde olabilir. Type 1, Type 2 isimli kötü huylu tümörler gibi..

Regression kısmında sınırsız sayıda obje durum varken classification kısmında sınırlı değerler vardır.

(2)

Unsupervised Learning find something interesting in unlabeled data. Sadece programa giriş(x) değerleri verilir. Ardından kendisine bu giriş verilerini değerlendirmesi ve ilgili şeyleri bulması istenir (benzerlikler ve farklılıklar).

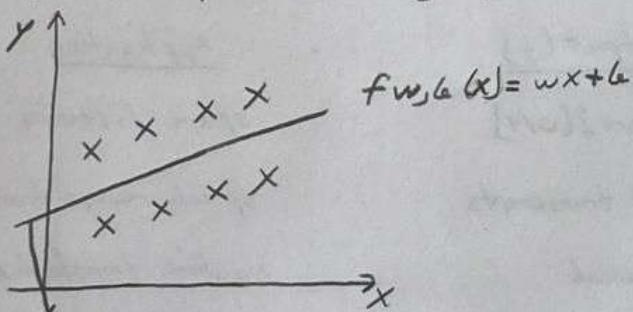
Bu süreç sonunda verileri farklı kümelere ayıracaktır. Bu clustering denir.

Clustering örneği: Google News her gün bir çok benzer haberin bir araya getirilmesi
 ↴
 Grouping customers

Linear regression model bir supervised learning örneğidir.

Bu model 'elinizdeki' veriye göre doğrusal eğriyi uydurur. Regression model numaraları tahmin eder. Bir evin boyutunu input olarak alıp fiyatını tahmin etmek gibi
 $x \rightarrow \text{input}$ $y \rightarrow \text{actual output}$ $\hat{y} \rightarrow \text{estimated output}$

How to represent f_p ? $\rightarrow f_{w,b}(x) = wx + b$



Bu model sızılık eğrisi esas olarak y değerlerini tahmin etmeye yarar. Doğrusal eğri yerine eğrisel, polinomyal kusacısı daha kompleks bir eğri de sızılebilir fakat linear regression en basit, en temel model olarak bilinir ve kolay uygulanır.

Univariate Linear Regression \rightarrow tek değişkenli girdiye verilen ismidir sadece evin (size)in varlığı

Cost function bütün machine learning modelleri için çok önemli bir kavramdır. Cost fonksiyonunun sonucu modelin ne kadar başarılı olduğunu anlamamiza yarar.

$f(x) = wx + b$ fonksiyonunda w ve b parametre olarak bilinir ve değerleri ile oynanarak en uygun doğrusal eğri bulunmaya çalışılır. En uygun doğrusal eğri bütün x değerleri için en yakın y değerini (y tahmini = \hat{y}) veren doğrudur. Bunu bulmak için cost (malzette) fonksiyonunu kullanıyoruz. Tahmin edilen y değerini gerçek y değerinden çıkarır ve sonra karesini alırız. Bu formülü her örnek için hesaplarız. $\underbrace{(\hat{y}_{ij} - y_{ij})^2}_{\text{error}}$

$$f_{w,b}(x^{(i)})$$

(3)

$J(w, b) = \frac{1}{2m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$ Bu fonksiyon m tane veri örneği için cost fonksiyonunu hesaplar. Fakat bu değer çok büyük olabileceği için m veri sayısına bölünür ve ortalamalı değer alınır. 2 ye bölünmesi de fonksiyon galisir fakat daha net sonuçları için genelde 2'e bölüyor. (Squared error cost function)

Farklı modeller için farklı cost function kullanılır fakat en çok tercih edilen budur. w değeri sıfırken w değeri ve $J(w)$ değeri 2 boyutlu düzlemede ifade edilebilir. Fakat w ve b sıfırdan farklı değerler için her ikisinin de etkisini hesaba katmak için 3 boyutlu düzleme ya da kontür grafik yöntemi ile minimum cost ($J(w, b)$) tespit edilebilir. Cost değerini minimalize etmek için tıpkı geleneksel gibi yerine "gradient descent" algoritması kullanılır. Bu algoritma basit yapıtlardan, derin öğrenme gibi çok karmaşık konularda da kullanılabilir. Bu algoritmanın temel mantığı; ilk olarak w ve b için 0 değerini verilir. Sonrasında algoritma w ve b değerlerini cost değerini minimum seviyeye getirmeye çalışır.

$$w = w - \alpha \frac{d}{dw} J(w, b), \quad b = b - \alpha \frac{d}{db} J(w, b)$$

learning rate (0-1)

w ve b değerleri eş zamanlı olarak güncellenebilir.

Eş zamanlı güncelleme yapabilmek için önce geçici değişiklik işlem yapıp sonra atama yapılır.

$$\text{tmp-}w = w - \alpha \frac{d}{dw} J(w, b)$$

$$\text{tmp-}b = b - \alpha \frac{d}{db} J(w, b)$$

$$w = \text{tmp-}w$$

$$b = \text{tmp-}b$$

Eğer learning rate denilen α değeri çok büyük seçilirse algoritma çok yavaş gelisir ve çok fazla zaman alır. Eğer learning rate çok küçükse çok büyük adımlar atar ve sürekli minimum noktasını kestiremez.

Eğer linear regression ile gradient descent formüllerini birleştirirsek, linear modelde minimum costu bulunuz bunun için cost fonksiyonun türevini gradient descente koyuyoruz.

$$w = w - \alpha \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)}) x^{(i)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})$$

Normalde birinden fazla yerel minimum değeri bulunabilir. Fakat linear regressionda sadece 1 tanedir.

(4)

Multiple features modeli tek tip girdi ile eğitmek yerine birden çok özellik ile eğiticiyi anaslar ve bu na göre matematiksel dayanak oluşturur.

$$f_{w,b}(x) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

example

$$f_{w,b}(x) = 0.1x_1 + 4x_2 + 10x_3 + -2x_4 + 80$$

size ↓ ↓ ↓ ↓
#bedrooms #floors years base price

$$f_{w,b}(x) = w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n + b$$

$\vec{w} = [w_1 \ w_2 \ w_3 \ w_4 \ \dots \ w_n]$, w parameters, satır vektörü

$$\vec{x} = [x_1 \ x_2 \ x_3 \ x_4 \ \dots \ x_n]$$

üstteki vektörler/ ifadeleri kullanarak multi features function'ı sadeleştirilebilir.

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b \quad (\text{bu yani multiple linear regression denir})$$

dot product

Vectorization hem bilgisayar CPU'su için hem de birleşik anlamda programlama işin en verimli yolların başında. Üst türündeki multiple linear regression model python'da şu şekilde ifade edilir

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

$$f = np.dot(w, x) + b$$

Feature scaling fonksiyonun parametrelerinin gerekce boyut değerlerine sahip x değerleri için (home size 300-2000) tıpkı değerler, göre boyut değerlerine sahip x değerleri için (#bedrooms) boyut değerler sepekti mantıklı sonuçlar verecek sistem geliştirmek. Fakat bu durum bazı dezavantajlar getirir örneğin parametrelerden bazıları çok büyükse ve diğerleri ona nazaran çok küçük kalırsa gradient descent global minimum bulmakta çok zorlanacaktır. Bunun için bütün parametreleri birer ya da aynı aralıklara rescale etmeniz gereklidir. Scale etmek için her bir değeri orijinal değerden kotsa da orjinalde kentler aralıklara verilen sıralısanır. Mean normalization ise değerleri genelde -1 ile 1 arasına scale eder. Z score ve mean z-score gibi yapılar da kullanılabilir. Burdaki aralık çok küçük ya da çok büyük olamalıdır. Feature scaling hiçbir durumda zarar verici bir durumda olmaz. Dolayısıyla herhangi bir şüphe durumunda direkt rescale yapmak garanti yoldur.

(5)

Önceki aşamalardaki işlemler yapıldıktan sonra gradient descent algoritmasının sonusuna gelip çalışmadığını辨别 etmek için bunun için J cost fonksiyonu ve iterations grafğini gözlemlayarak convergence olup olmadığını anlayabiliriz ya da kırık bir ϵ (epsilon) değeri sayesinde J cost fonksiyonunun azalması ile karşılaştırırız ve buna göre convergence durumunu ~~elde ettiklerimiz~~.

Learning rate değeri çok büyük segilidigi zamanlarda çok fazla iterations ile karşılaşır. Learning rate değeri büyük segilidigi zamanlarda cost fonksiyonunu ya sürekli artarken ya da zaman zaman artıp zaman zaman artarak görevliliğin bu aynı zamanda hatta olduğuna da işaret edebilir.

Feature engineering isimli bir kavram vardır ve bu kavram aslında elinizde olan original features'i kullanarak yeni feature oluşturma olayıdır. Örneğin bir ev arasının dikdörtgen olduğumu varsayırsak ve uzun ve kısa kenarı kullanarak oluşturmuş bir model yerine uzun ve kısa kenarı separata alan bulutuz ve yeni bir feature elde etmiş oluyoruz eski den kullanıldığından 2 katelli bir model yerine feature engineering kullanarak 3 katelli bir model kullanmak daha doğru sonuçlar veren bir model ortaya koyma.

Polynomial regression kısmında x^2 ya da x^3 lü ifadeler katarak eğrisel bir yapı oluşturmak istenir ve bunun neticesinde daha iyi modeller oluşturulabilir. Fakat burada doğrulukla en büyük terim x^2 gibi üstü ifadeler yerine x^3 tarzı üstü ifadeler kullanılmıştır. Bunun sebebi x^2 li grafiklerin U tarzında bir sekilde sahip olmaları ve x ekseniindeki değer artarken tan tersi bir rol arayışına gitme yönelmeleridir. Bunun yerine x^3 lü (\sim) bir denklem bütün x değerleri için bir artış yoluna tabi olur.

$$f_{w_1, w_2}(x) = w_1 x + w_2 x^2 + w_3 x^3 + b \quad \text{örnek verilen fonksiyonda aynı } x \text{ değeri } 5 \times 10^{-3} \text{ ve } 1 - 10^6 \text{ olmasının üstü ifadenin dahil olması ile range (ranget)}$$

olduğu anlaşılır. Bundan dolayı feature scaling uygulamak gereklidir. Bir aralık elde etmek için önceliği olacak \sqrt{x} li ifadeler (\sim) kullanılabilir fakat hiçbir zaman yatay hale gelmez, sürekli bir artış durumunu içindedir. Bu ifadelerin yapılmasıında scikit learn kütüphanesi kullanılabilir.

(6)

Classification genel olarak True or False, 1 or 0, Yes or No gibi yapıda y output değerlerini sınıflandırma oldukça iyi parçalara genelde binary classification yapıyoruz. Bu tane sınıflandırma problemlerinde linear regression yetersiz olduğu için logistic regression kullanılabilir.

0 ya da 1 outputu istenen bir problemdede sigmoid(logistic) function kullanılır.

$$\frac{1}{1+e^{-z}}, \quad z = \vec{w} \cdot \vec{x} + b, \quad \text{for } z=0 \text{ outcome } \frac{1}{2}$$

for $z > 0$ outcome ≥ 1
for $z < 0$ outcome ≤ 0

$f_{\vec{w}, b}(\vec{x}) = 0.7$, 0 ile 1 arasında 0.7 değerini alır bunun %70 olasılık率为 1 olasılığıdır. Farklı notasyonlarda ifade edilebilir $P(y=0) + P(y=1) = 1$

$f_{\vec{w}, b}(\vec{x}) = P(y=1 | \vec{x}; \vec{w}, b)$ refers to probability that y is 1, given input \vec{x} , parameters \vec{w}, b

Linear regression'da cost fonksiyonundaki ifade yerine logistic regression'da loss function kullanıyoruz.

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m L(\underbrace{f_{\vec{w}, b}(\vec{x}^{(i)})}_{\text{loss}}, y^{(i)})$$

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) - (1-y^{(i)}) \log(1-f_{\vec{w}, b}(\vec{x}^{(i)}))$$

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m [(-y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1-y^{(i)}) \log(1-f_{\vec{w}, b}(\vec{x}^{(i)})))]$$

Linear regression $\rightarrow f_{\vec{w}, b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

Logistic regression $\rightarrow f_{\vec{w}, b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w} \cdot \vec{x} + b)}}$

use logistic regression formula in the cost function.

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right]$$

$$b = b - \alpha \left[\frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) \right]$$

Eğer logistic regression türü mesela linear regression eğitimi setine iyi uyum gösteremiyorsa (does not fit well) bu da underfitting denir ya da algorithm has high bias disponibiliti.

Fakat herhangi bir function eğitimi setine çok iyi ayak uyduruyorsa error neredeyse 0 ise bu da overfit denir ya da algorithm has high variance.

algorithm is trying very hard to fit every single training example.

(7)

bulunabilecek en iyi model underfit veya overfit olmamalıdır. Bir diğer deyişle high bias ve high variance durumlarından kaçın olmalıdır yada farklı değerlerin tahmin etmektede çökmemek için çok zorlanırız. Overfit durumunu aşmanın çeşitli yolları vardır örneğin eğitim seti'ne yeteneğe ve çok fazla feature varsa overfit olusabilir. Bundan dolayı ya feature sayısını azaltmamızı ya da eğitim setini arttırmamızı. Diğer yol ise reduce size of parameters denilen olay yani "regularization" dir.

$$\text{Regularization} \rightarrow \frac{1}{2m} \left[\sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \underbrace{\lambda \sum_{j=1}^n w_j^2}_{\text{"lambda" regularization parameter}} + \underbrace{\frac{\lambda}{2m} b^2}_{\text{regularization term}} \right] \xrightarrow{\text{can include or exclude}}$$

$$\text{But easiest and simpler one} \rightarrow \min_{\vec{w}, b} \left[\frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2 \right]$$

Eğer lambda 0 veya 0'a yakındır bir değer seçersek overfit ile karsılıklı. Lambda'ı çok büyük bir değer olarak seçersek de underfit ile karsılıklı.

Gradient descent

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b) \rightarrow \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} w_j$$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b) \rightarrow \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) \quad (\text{don't have to regularize } b)$$

Yukarıdaki iki formül hem linear hem de logistic regression için geçerlidir fakat unutulmamalıdır ki f in tanımı linear regression ve logistic regression'da farklı doklärılıyla o türslere diktat edilmeli.

Advanced Learning Algorithms

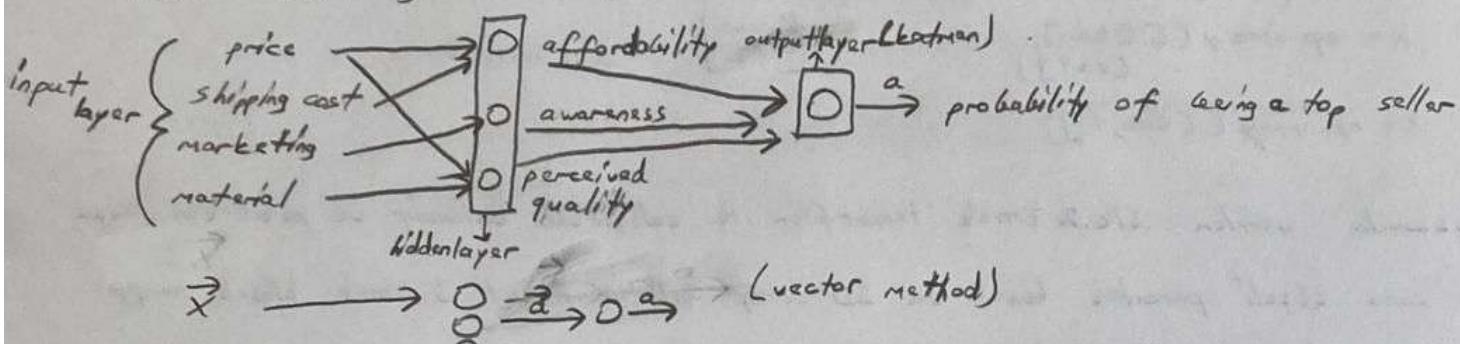
Bu kurs nöral ağlar ve derin öğrenme algoritmalarını esas alır. Decision tree'ye de denir. Bunlar yaygın olarak (en çok) kullanılan algoritmalardır. Bu kursta nasıl makine öğrenmesi uygulanması ve nasıl çalıştığı hakkında pratik tavsiyeler de yer alır.

- Neural Networks
 - inference (prediction)
 - training
- Practical advice for building machine learning systems
- Decision trees

Nöral ağlarla alakalı ilk çalışma 1950'lerde insan beynini tasvir eden bir yazılım ya da algoritma yapmıştır. Genel itibarı ile 1950'lerde çalışmalarla başlandı, 1980 ve 1990'ların başında kullanıldı, fakat 1990'ların sonlarında gözden düşüldü. 2000'den sonra ise tekrar yükselişe geçti. Nöral ağlar genel olarak insan zihniyi tasvir etme ya da kopyalama amacıyla kullanılsa da bugün itibarıyla insan zihni hakkında çok fazla bilgimiz yok ve her geçen gün yeni şeyler öğreniyoruz. Dolayısıyla bu çalışmalar daha doğru sonuçlar almak için mühendislerin kullandığı bir modeldir. Derin öğrenmenin ve bu tür çalışmaların gündeme gelmesideki en önemli sebeplerinden biri internet ve teknolojik aletlerin yaygınlaşması, sayesinde sahip olduğumuz veriseti birçok alanda arttı. Aradan bu voi nüktelerin rağmen linear regression ve logistic regression gibi geleneksel AI algoritmalarının performansında beklenen olumsuz sonucun ortaya çıkmaması dolayısıyla bu alanda çalışan insanlar yeni bir bireşim açısı geliştirmek istediler ve bu nöral ağlarıdır. Nöral ağlar sayesinde varlığı artırılmış performans artışı yakalabildi. Derin öğrenme konusunda önemli olan konulardan biri ise son zamanlarda eğitsayar işlenmesi ve GPU gibi dersinsel yapının gelişmesi olması ve image recognition'dan data eğitsile alana kadar data tabii telli işe sunabilmesidir.

(9)

Tane içinde nöral ağlar satıcı olduğumuz varlığı ile yeni bir bilgi forcusu ve bunu istenen sonucaya kadar gösterme işidir.



Satıcı olduğumuz inputlar yeni veriler elde ederek outputa ulaşmaya çalışır. Burda tek değer ya da 2 değer listesiyle sonucaya ulaşır, fakat yukarıdaki yapı denkisi 4 input kullanarak da bir feature belirleme能力和ı üzerinde sorun olan kisim bu tarz bir modelde tek tek katmanlar arasındaki bağlantıları düşünmek ve ona göre grupları kurmak. Bu durum çok nüfuzlu olduğu için her bir girdi sonraki katmandaki bir nöronlara gidiyormuş gibi diperdiler ve ardından bir vektör belirleyerek işler yapılır. Input layer ve output layer ıçın aktivasyon işlemi önemlidir, fakat ara katmanlarda aktivasyon değerleri genel bir anlam ifade etmediği için hidden layer denir. Yani ikişim ıçın (x, y) değerlerinin tutarlı olması önemlidir. Hidden katmandaki özellikler feature engineering kullanılarak elde edilir. Farklı sayıda hidden layer'a sahip modeller genellikle daha geniş ağdan beklemeye epüllü olabilirler. Örneğin face recognition problemlerde pixel olarak girdiler alınır ve outputa yaklaştıkça fotoğrafın türkiz parçaları esas alınarak değerlendirilir.

Bir katmandaki nöronun aktivasyon değerini bulmak ıçın bir önceki katmanın sonucları bilinmelidir ve bu durum tahmin etme veya antren etmeye işine de yarar.

örnek

$$\begin{array}{c} \xrightarrow{x} \\ \text{input} \end{array} \xrightarrow{\text{layer 1}} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\text{layer 2}} \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \\ a_3^{[2]} \end{bmatrix} = g(\vec{w}_1^{[2]}. \vec{a}^{[1]} + b_1^{[2]})$$

genel formül $\Rightarrow a_j^{[2]} = g(\vec{w}_j^{[2]}. \vec{a}^{[1]} + b_j^{[2]})$

(parameters w and b of layer 2, unit j)

Tensorflow derin öğrenme ıçın geliştirilmiştir. Google tarafından Keras ile birleştirilip yeni bir sürümleri yayınlanmıştır. Genellikle Keras arayüzü kullanılır.

(10)

Tensorflow ve Numpy ile çalışmakta bazı farklılıklar vardır:

$x = np.array([200, 17])$ $\begin{bmatrix} 200 \\ 17 \end{bmatrix}$ 1×2 matrix

$x = np.array([[200], [17]])$ $\begin{bmatrix} 200 \\ 17 \end{bmatrix}$ 2×1 matrix

$x = np.array([200, 17])$

Yukarıda verilen ilk 2 örnekle tensorflow ile çalışırken kullanılır ve yazılırken epey ve sonra köşeli parantez konur bu 2D array belirtir. Fakat 3. örnekte numpy array tanımlıdır ve 1D array belirtir sayısının listesini tutar. Tensorflow'un 2D arrayde çalışmasının sebebi genel olarak çok büyük veri setleri ile işlem yapıldığı için programı daha verimli çalıştırma yaratır.

örnek neural network kodu

$x = np.array([200.0, 17.0])$

$layer_1 = Dense(units=3, activation="sigmoid")$

$a1 = layer_1(x)$

$layer_2 = Dense(units=1, activation="sigmoid")$

$a2 = layer_2(a1)$

Bu kod yazımı zor ve gittikçe karmaşık bir yapıdadır.

Daha basit yazımı model oluşturmak için \rightarrow $layer_1 = Dense(units=3, activation="sigmoid")$
 $layer_2 = Dense(units=1, activation="sigmoid")$

farklı yazım
 $model = Sequential([$
 $Dense(units=3, activation="sigmoid")$
 $Dense(units=1, activation="sigmoid")])$

$x = np.array([[[200.0, 17.0],$
 $[120.0, 5.0],$
 $[425.0, 20.0],$
 $[212.0, 18.0]]])$

$y = np.array([1, 0, 0, 1])$

$model.compile(...)$

$model.fit(x, y)$

$model.predict(x_new) \rightarrow$ en son programda yeni verileri tahmin ettiğiniz.

AI konusunda çok geniş bir alanda kapsar ve farklı başlıklarla katılabılır.

AI

ANI

(Artificial narrow intelligence)

E.g., smart speaker, self-driving car
 web search, AI in farming and factories

AGI

(Artificial general intelligence)

Do anything a human can do

Nöral ağların kodunda bir klasik forward propagation kodu vardır. Bu de vectorization

Kodu vardır For loops

```
x = np.array([[200, 17]])
W = np.array([[1, -3, 5],
              [-3, 4, -6]])
```

```
b = np.array([-3, 12])
```

```
def dense(A-in, W, b):
```

```
    units = W.shape[1]
```

```
    A-out = np.zeros(units)
```

```
    for j in range(units):
```

```
        w = W[:, j]
```

```
        z = np.dot(w, A-in) + b[j]
```

```
A-out[j] = g(z)
```

```
return A-out
```

$[1, 0, 1]$

*** epochs = number of steps in gradient descent.

Nöral ağlarda önceden öğrendiğiniz loss function compile fonksiyonu ile sağlanır ve binary classification problemlerinde işe yarar.

handwritten digit classification problem \rightarrow binary classification

$$L(f(x), y) = -y \log(f(x)) - (1-y) \log(1-f(x))$$

compare prediction vs. target

Tensorflow'da loss function yerine binary cross entropy ismi ile bilinir.

Son olarak gradient descent algoritması ile minimum değeri bulmak için tespit edilmiş algoritmanın fonksiyonu Tensorflow'da mevcuttur. Aynı zamanda backpropagation yapar.

model.fit(X, Y, epochs=100)

\downarrow
100 kez

compute derivatives for gradient descent using
"back propagation"

Sıra ana kodda aktivasyon fonksiyonu olarak sigmoid function kullandık fakat bu da durumlarında değer aralığını değiştirmek isteyebiliriz. Bu gibi durumlarda linear activation function, ReLU, softmax activation gibi fonksiyonlar da tercih edilir ve zaman zaman sık rastlanır şekilde nöral ağ yapısının kalitesini artırır.

Farklı birçok activation function olmasına rağmen buna rağmen hangisini kullanacağınızda karar verme problemi olabilir. Bu gibi durumlarda output kismi öncelik olabilir hatta her bir kısımında hidden layer düşünerek de karar verebiliriz. Örneğin output y_1 değeri 0 veya 1 olan problemleri binary classification olarak değerlendirdiğimiz için sigmoid function data mantıklı olacaktır. Fakat regression problemi ikililikten yoksak farklı bir activation function mantıklı olacaktır. Örneğin bir şirketin dikkat ile bugünkü hisse değerlerini tahmin edebesek linear activation function data mantıklı olabileceğini negatif ve pozitif değerler alıyor. Fakat evdeki hisse değerini hep pozitif olacağı için ReLU activation en mantıklı seçenek olurdu. Hidden layer kismi için ise en çok ReLU activation tercih ediliyor. ReLU function, sigmoid function'a göre data hızla denebilir. İkinci sigmoid function'da exponential işlemi ve negatif işlemi yapılması ikinci seçenek ise ReLU function sadece 0'dan başıktan flat durumunda sigmoid function ise hem $+\infty$ hem $-\infty$ giderken flat durumunda bu nedenle gradient descent algoritmasını yavaşlatır. Özellikle hidden layerda linear function veya ReLU data mantıklıdır. Bunun sebebi ve α değerleri grafikte eğriliği değiştirmek ve bunu sonraki katmanlarda etkilemek dolayısıyla sürekli farklı eğriliği linear function kuramızı生成etir. ReLU'da ise eğri değişen yerlerde 0'dan büyük olana kadar 0 değerini alıyor.

Multiclass classification problem y değerlerinin 2'den fazla değer sahip olmasıdır. Mesela 0 ve 1 tespiti yerine tam rakamları tespit etme problemi gibi softmax algorithm genelde 2'den fazla sınıfa sahip problemleri çözümlemekte işe yarıyor.

Softmax regression (4 possible outputs)

$$z_1 = \vec{w}_1 \cdot \vec{x} + b_1 \quad \alpha_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=1 | \vec{x})$$

$$z_2 = \vec{w}_2 \cdot \vec{x} + b_2 \quad \alpha_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=2 | \vec{x})$$

$$z_3 = \vec{w}_3 \cdot \vec{x} + b_3 \quad \alpha_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=3 | \vec{x})$$

$$z_4 = \vec{w}_4 \cdot \vec{x} + b_4 \quad \alpha_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=4 | \vec{x})$$

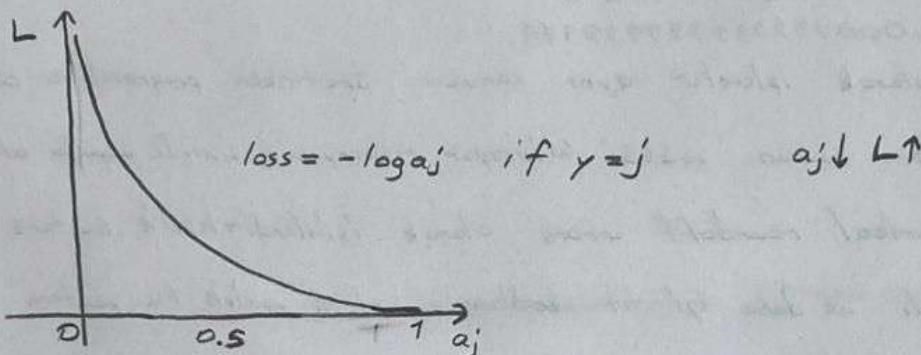
(13)

softmax regression is the generalization of logistic regression.

softmax regression

$$\text{loss}(a_1, \dots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y=1 \\ -\log a_2 & \text{if } y=2 \\ \vdots \\ -\log a_N & \text{if } y=N \end{cases}$$

Crossentropy Loss



Bu grafik y degerindeki ay degerinin yani olasılığın ya da regresión kuralı uygulanıp eton sonra ortaya çıkan degerin kışılıklıya Loss degerinin bıyadığı anlamına gelir.
Daha önceki yaptığımız 0 ve 1 sayısının eğitiminde 25-15-1 units şeklinde
bir nöral ağı tasarlamıştık ve output layerda logistic regression kullandık. Eğer
bu 0 dan 1'a kadar olan rakamları eğitmek ve bu şekilde bir sınıf almak istiyorsak
25-15-10 units şeklinde bir tasarım yapabiliriz ve 10 farklı çıktıya (output layer)
sahip olmak istiyorsak bu noktada logistic regression yerine softmax regression kullanılır.

Tensorflow code with softmax:

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
    Dense(units=25, activation="relu"),
    Dense(units=15, activation="relu"),
    Dense(units=10, activation="softmax")
])
```

```
from tensorflow.keras.losses import SparseCategoricalCrossentropy
model.compile(loss=SparseCategoricalCrossentropy())
model.fit(X, Y, epochs=100)
```

Note: It is not the best version of code
Better version later

(14)

Bir önceki sayfadaki code doğru çalışmamasına rağmen bazı istisnalar vardır ve bunların getirebileceği durumlardan kurtulmak için data iyi bir kodlama yöntemi gereklidir.

Örneğin: $x_1 = 2.0 / 10000$
 $\text{print}(f\{"x_1: .18f\})$

Sonuç: 0.00020000000000000000

$x_2 = 1 + (1 / 10000) - (1 - 1 / 10000)$
 $\text{print}(f\{"x_2: .18f\})$

Sonuç: 0.00019999999999978

Matematiksel olarak işlemler aynı sonucu üretirken programlama dili farklı sonuçlar üretiyor bunun sebebi bilgisayar memorysinin sınırlı yapıda olmasıdır. Ayrıca bu olayı numerical roundoff errors olarak isimlendirmektedir. Bu tür hataları önlemek için data farklı ve data iyi bir kodlama yöntemi vardır. Bu yöntem numerical roundoff errors değerini azaltır ve data doğru hesaplama yapmanıza yardımcı olur.

Softmax Regression

$$[a_1, \dots, a_{10}] = g(z_1, \dots, z_{10})$$

$$\text{Loss} = L(\vec{a}, y) = \begin{cases} -\log a_1 & \text{if } y=1 \\ \vdots \\ -\log a_{10} & \text{if } y=10 \end{cases}$$

More Accurate version

$$L(\vec{a}, y) = \begin{cases} -\log \frac{e^{z_1}}{e^{z_1} + \dots + e^{z_{10}}} & \text{if } y=1 \\ \vdots \\ -\log \frac{e^{z_{10}}}{e^{z_1} + \dots + e^{z_{10}}} & \text{if } y=10 \end{cases}$$

Code of more accurate version:

model = Sequential([

Dense(units=25, activation='relu'),

Dense(units=15, activation='relu'),

Dense(units=10, activation='linear'),

model.compile(loss=SparseCategoricalCrossentropy(from_logits=True))

Bunun sebebi 'ara işlemleri' kaldırarak ve tensorflow işlemlerinden yapsası için itin vermektedir. Bu kodu bütünyle logistic regression için yazmak isterseniz dikkat edilmesi gereken birkaç yer vardır.

logistic regression
(more numerically accurate)

model → model = Sequential([

Dense(units=25, activation='sigmoid'),

Dense(units=15, activation='sigmoid'),

Dense(units=1, activation='linear')])

from tensorflow.keras.losses import BinaryCrossentropy

loss → model.compile(..., BinaryCrossentropy(from_logits=True))

model.fit(X, Y, epochs=100)

fit $\rightarrow \text{logit} = \text{model}(x)$
predict $\rightarrow f(x) = \text{tf.nn.sigmoid(logit)}$

$\text{tf.nn.softmax(logit)}$ for softmax algorithm

probability değerlerini almak için bu kod kusmaları kullanılır

Şu anda koder multiclass classification problemini ve çözümünü gördük. Bunun dışında birde multilabel classification problem vardır.

Örnegin elinizde 3 farklı fotoğraf olsun ve yapay zeka su soruları cevaplasın.		
	1. foto	2. foto
Is there a car?	yes	no
Is there a bus?	no $y = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$	yes $y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$
Is there a pedestrian?	yes	yes

Bu problemi çözmek için 3 tane ayrı nöral ağ yapısı oluşturulup tek output sigmoid function kullanılarak belirlenebilir ya da 3 output üzerinden nöral ağ kırılımların outputa sigmoid function uygulanır ve sonucu gözetlenir.

* Genel olarak sıkılıkla kullandığımız gradient descent algoritmden daha iyi salıpçı algoritmalar vardır nöral ağında kullanmak için bu algoritmaları inceleyeceğiz.

Gradient descentte bir alpha değeri belirtilir ve minimum costa ulaşana kadar sürekli aynı alpha değerde adım atar fakat bu alpha değeri otomatik olarak ve data tabuk sonuca ulaşın isteyebiliriz bunu "Adam" algoritması denir. Eğer a learning rate değerini olması gerekenden büyük seviyelerde salınan (oscillating) bir yapıda olur ve learning rate değerini kısıtlamak isteyebiliriz. "Adam" algoritmi bununla da başa gider.

Adam: Adaptive moment estimation

compile komisyonuna dahil ederiz geri tabanlar ayndır.

```
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 1e-3),
               loss = tf.keras.losses.SparseCategoricalCrossEntropy(from_logits = True))
```

*** Python'da türde işlemlerini yapmak için sympy isimli bir kütüphane kullanılabilir.

* Forward propagationdan sonra backward propagation yapılabılır. Backward prop'un söyle bir okunuşu yani varsa bir matematiksel ifadesi parçalara ayırdıysak mesela $J = \frac{1}{2} d^2$ olsun
 $d = a - y$, $a = c + b$, $c = wx$ gibi amaciımız forward prop ile J_y 'yi bulmaktır.

Eğer back prop yaparsak verdığımız örnekte her bir parametrenin en sondaki J' 'yi nasıl etkilediğini öğrenmemize yardımcı olur (Pascal's rule ve chain rule ile)

→ Machine learning toolarını öğrenmekten sonra gelen önemli kisim bunları kullanabilmek hayatda uygulayabilmektir. Regularized linear regression uyguladığınız bir problemi düşünelim housing price gibi anlıkende kabul edilemeyecek seviyede büyük bir hata ile karşılaşırız. (Tabii ki yaptığınızda)

Yapabileceğiniz bazı temel adımlar vardır:

1. Daha fazla eğitim ömrü kullanmak
2. Daha az feature'a sahip bir uygulama denemek
3. Elestra feature edinmeyi denemek
4. Polynomial feature eklemeyi denemek (x_1^2, x_2^2, x_1x_2 , etc)
5. λ değerini azaltmak veya artırmak

Bu ve bu maddeler daha iyi bir model için kullanılabilir. Fakat bu teknikle hangi maddenin bir istedığınız sonuca götüreceğini tanısını koymak oldukça fazla zaman ^{saglayıcı} alır. Bir model saglayıcının ne kadar az error değeri elde ederse o kadar iyi bir model saglayıcının dışarıda mesela 1.derece denklemlen 10.derece denkleme kadar modelleriniz olsun ve hangisi dada az error verirse onu seçmek daha mantıklı gelebilir. Fakat bu noktada d (degree of polynomial) değeri test sete göre seçildiği için olması gereken genelleme hatalarından dada az hata veren eğitimindeki. Bunun yerine en iyi modeli otomatik olarak seçmek için farklı bir yol vardır. Normalde elinizdeki veri setini eğitem ve test set olarak ayırmak yerine ⁴⁶⁰₄₆₀ ²⁰₂₀ ¹⁰₁₀ ⁵₅ ³₃ ²₂ ¹₁ test set olanağı 3'e ayıriz. Bundan sonra ilk denekdeki gibi tetkiklerin fakat test set yerine cross-validation set kullanarak uygularız. Örneğin 1.dereceden 10.dereceye kadar 10 farklı polynomial modelleriniz olsun ve eğittiğinden sonra cross-validation set ile test edelim ve en az error veren modeli seçelim. Ardından son olarak modelin ne kadar iyi olduğunu anlamak için test set ile test edelim. Bu belli bir eşittir ayni zamanda nöral ağlarda farklı tesirlerin farklı hidden layer sayısı ve layerleri ^{sayısı} sayesinde de hangisinin dada iyi olduğunu söylemeye yarar.

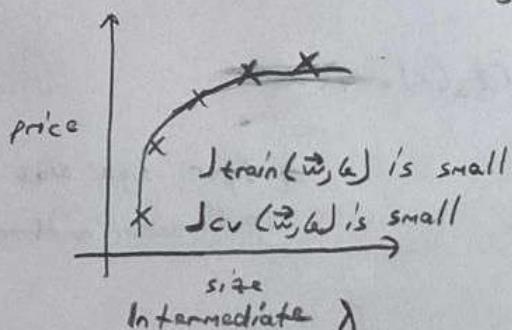
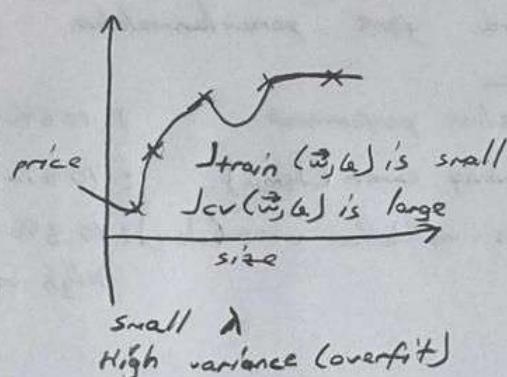
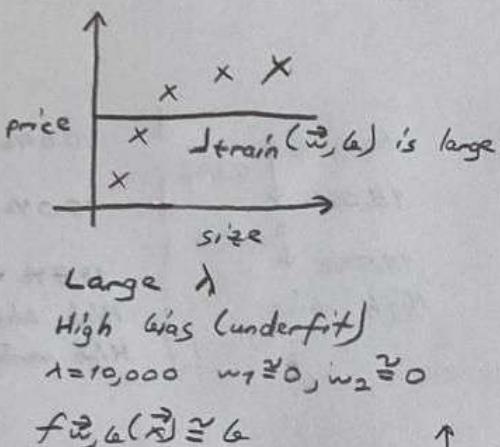
(17)

High Bias (underfitting) ve high variance (overfitting) kavramlarını data once görmüştik. Her iki durum da size yanıtız. Overfit mi underfit mi içindiriz veya bu durumları nasıl düzelttiğimiz sorusuna cevap polynom derecesini değiştirmek birlikte regularization termde düşündürmekle.

Linear Regression with regularization

$$\text{Model: } f_{\vec{w}, b}(\vec{x}) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$$

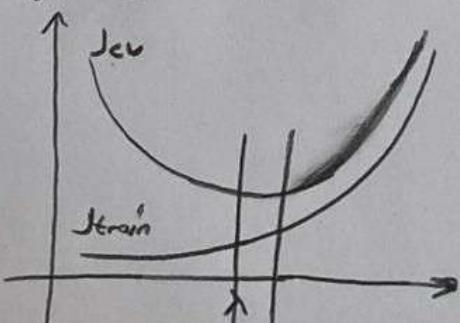
$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$



Choosing the regularization parameter λ

1. Try $\lambda = 0 \rightarrow \min_{\vec{w}, b} J(\vec{w}, b) \rightarrow \vec{w}^{(1)}, b^{(1)} \rightarrow J_{\text{cv}}(\vec{w}^{(1)}, b^{(1)})$
2. Try $\lambda = 0.01 \rightarrow \vec{w}^{(2)}, b^{(2)} \rightarrow J_{\text{cv}}(\vec{w}^{(2)}, b^{(2)})$
3. Try $\lambda = 0.02 \rightarrow \vec{w}^{(3)}, b^{(3)} \rightarrow J_{\text{cv}}(\vec{w}^{(3)}, b^{(3)})$
4. Try $\lambda = 0.04$
5. Try $\lambda = 0.08 \rightarrow J_{\text{cv}}(\vec{w}^{(5)}, b^{(5)})$
- ⋮
12. Try $\lambda \approx 10 \rightarrow \vec{w}^{(12)}, b^{(12)} \rightarrow J_{\text{cv}}(\vec{w}^{(12)}, b^{(12)})$

örnekte olduğu gibi kelebekli λ değerlerini deneyerek en iyi değeri seçeriz. (en az hata)



the intermediate value of lambda is the best

Bias ve variance kavramlarını kullanırken high ve low gibi kelimelerle kullanıyoruz. Fakat neye göre high veya low dediğiniz önemli bir noktadır örneğin bir speech recognition problem düşünülmeli. Bu problemede training set hatası olarak %10.8 lik bir değer alırsak bunu kekece yetersiz (high) bir değer denebilir. Burda önemli konu pudurki voice (sesin)ensusut olma durumu vardır yani konusmacı %10.6 lik bir hata ile konuşuyor (antepitroyan telaffuz ya da genre sistem noise). %10.6 ile %10.8 arasında çok büyük bir fark vardır ve artık konu kekece faktörlere göre yorumlanmalıdır.

Örnek

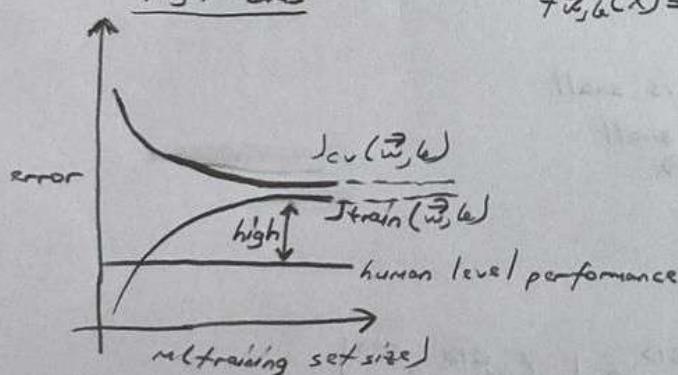
Baseline performance	: 10.6%	\uparrow	0.2%
Training error (J_{train})	: 10.8%	\downarrow	4.0%
Cross validation error (J_{cv})	: 14.8%	\downarrow	High variance

10.6%	\uparrow	4.4%	10.6%
15.0%	\downarrow	0.5%	15.0%
15.5%	\uparrow	4.7%	19.7%
High bias			High bias
			High variance

** Learning Curves:

High Bias

$$f_{\theta, \omega}(x) = w_1 x + b$$

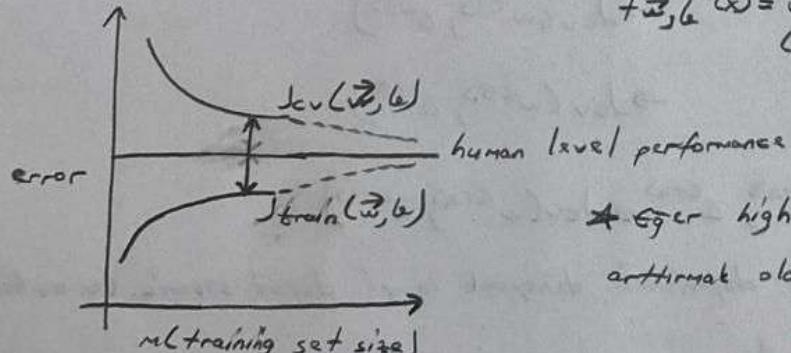


* Eger high bias bir modelimiz varsa eğitimi setini artırmakla ugrasınca zaman kaybeder.

High variance

$$f_{\theta, \omega}(x) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$$

(with small λ)



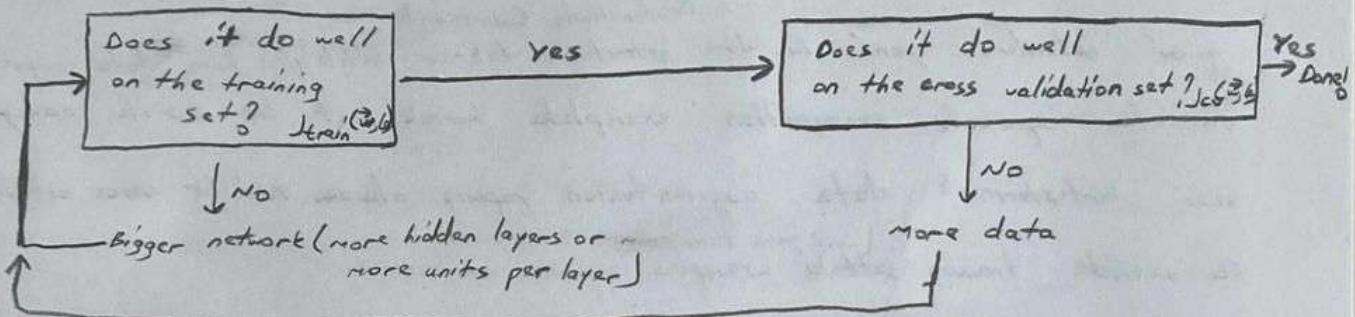
* Eger high variance bir modelimiz varsa eğitimi setini artırmak olduktan sonra olabilir.

Bu konuya ilgili 6 farklı modelde yaygınlaşılmış ve bu modeller overfitting ve underfitting durumunu düzeltmek için hangi modelde hangi durumdan kurtulmanıza yardımcı olur?

- Get more training examples → fixes high variance
- Try smaller sets of features → fixes high variance
- Try getting additional features → fixes high bias
- Try adding polynomial features (x_1^2, x_2^2, x_1x_2) → fixes high bias
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

** Neural Networks and bias variance

Large neural networks are low bias machines.



Gittikçe daha büyük bir nöral ağ yapmak daha iyi bir sonuçu verir yoksa bir süre sonra high variance'a mi düşer?

Model ayırdıkça regularized metodunu kullanırsak model en kötü ihtimale aynı seviyede kalır ya da daha iyi bir model olur. Vani 30-30-30-1 gibi bir nöral ağ 10-10-1 bir nöral ağdan genellikle daha iyi performans gösterir.

Tensorflow code

Unregularized MNIST model

```

layer-1 = Dense(units=25, activation='relu')
layer-2 = Dense(units=15, activation='relu')
layer-3 = Dense(units=1, activation='sigmoid')
model = Sequential([layer-1, layer-2, layer-3])
  
```

Regularized MNIST model

```

layer-1 = Dense(units=25, activation='relu',
                 kernel_regularizer=L2(0.01))
layer-2 = Dense(units=15, activation='relu',
                 kernel_regularizer=L2(0.01))
layer-3 = Dense(units=1, activation='sigmoid',
                 kernel_regularizer=L2(0.01))
model = Sequential([layer-1, layer-2, layer-3])
  
```

(λ)

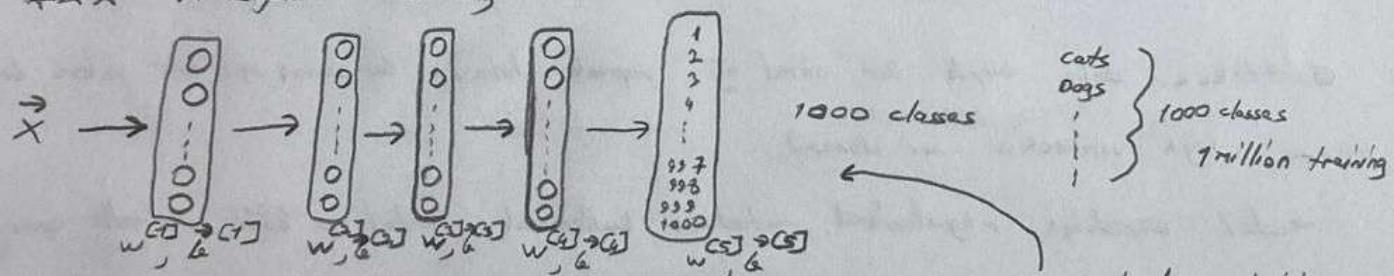
Makine öğrenmesi sürecinde adımların hangi sırayla yapılacağına dair bir loop oluşturabiliriz. Bir makine öğrenmesi projesinde error analysis önemsiz bir yer taşır. Mesela 1000 örnekten oluşan bir uygulama olsun ve 100 örnek yanlış sınıflandırılmış olsun. Bu 100 yanlış sınıflandırılmış örnekten en çok hangi etikette ya da alanda hata varsa oncelikle ona uygun daha mantıklı olur. Bu noktada da bu hatayı azaltmak için veri toplanır. Bu yolların birinden Farklı veri toplanarak yine yapılabilir bazı şayler vardır:

Bunlar aynı görseli:

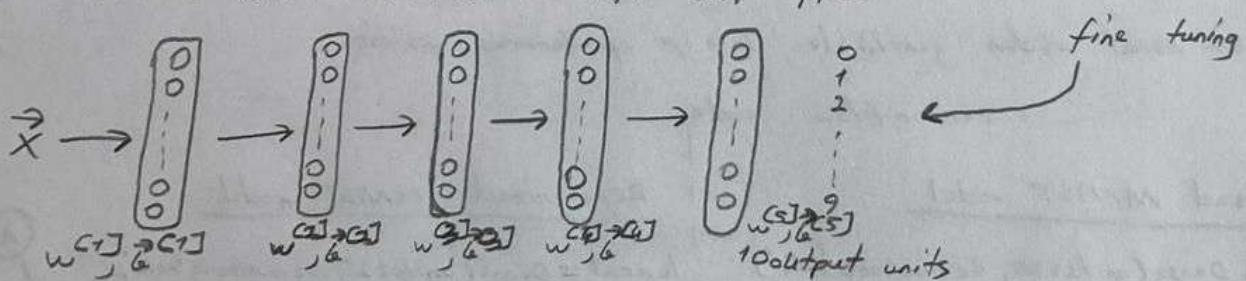
1. Bölgelerde
2. Küçükler
3. Rotate edenek
4. Renkini değiştirmek?
5. Arka plana gürültü (noise) eklemek
6. Distortions (warping)

gibi işlemlerle yeni eğitimlere getirilerek tekrar eğitilebilir. Buna "data augmentation" denir. Mesela speech recognition exampleda kerrak, net bir ses ile noisy background sesi birleştirerek data augmentation yapmış oluruz. Yeni bir voice ortaya çıkar. Bu şekilde training setindeki examples sayısını artırırız.

*** Transfer Learning



İstediğimiz neural networkin son layer haric kopyala.



option 1: only train output layers parameters. (smaller sets of data)

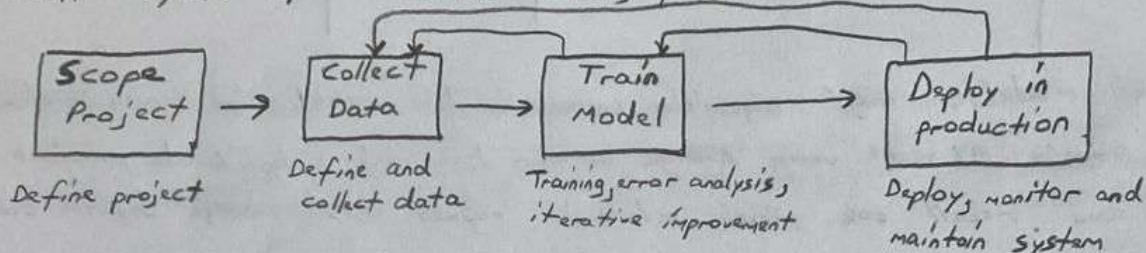
option 2: train all parameters. (a little bit longer data work better)

İlk nöral ağı 1000 farklı sınıfa sahipken output layer dışındaki parametrelerin 0 ile 9 arası arasındaki sayıları tespit eden bir yeni nöral ağıda buttonluksu detect edges - detect corners - detect curves/basic shapes gibi operatörleri çok fazla örmekle test etmesi ve bunu nasıl yapacağımız öğrenmesidir.

Two steps for Transfer Learning:

1. Download neural network parameters pre-trained on a large dataset with same input type (e.g., images, audio, text) as your application (or train your own).
2. Further train (fine tune) the network on your own data.

Full cycle of a machine learning project



Precision / Recall

$y=1$ in presence of rare class we want to detect

		Actual Class	
		1	0
predicted class	1	True Positive 15	False Positive 5
	0	False negative 10	True negative 70

Precision: (of all patients where we predicted $y=1$, what fraction actually have the rare disease?)

$$\frac{\text{True Positives}}{\# \text{predicted positive}} = \frac{\text{True Positives}}{\text{True pos} + \text{False pos}} = \frac{15}{15+5} = 0.75$$

Recall: (of all patients that actually have the rare disease, what fraction did we correctly detect as having it?)

$$\frac{\text{True Positives}}{\# \text{actual positives}} = \frac{\text{True Positives}}{\text{True pos} + \text{False neg}} = \frac{15}{15+10} = 0.6$$

Trading off precision and recall:

Logistic regression: $0 < f_{\theta, b}(x) < 1$

Predict 1 if $f_{\theta, b}(x) \geq 0.5$

Predict 0 if $f_{\theta, b}(x) < 0.5$

Suppose we want to predict $y=1$ (rare disease) only if very confident.

threshold value = 0.7, 0.9, ... bigger than 0.5

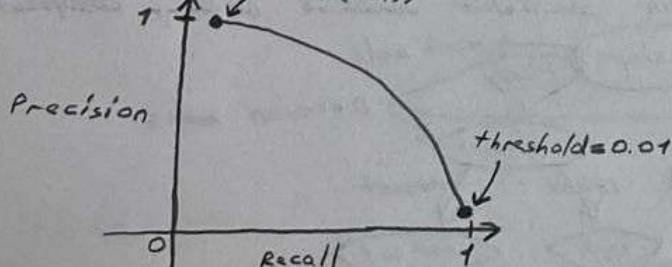
higher precision, lower recall

Suppose we want to avoid missing too many case of rare disease (when in doubt predict $y=1$)

for example threshold value is 0.3

lower precision, higher recall

threshold = 0.99



* * * F1 score:

	Precision (P)	Recall (R)	$\frac{P+R}{2}$	Average	F1 score
Algorithm 1	0.5	0.4	0.45	0.444	
Algorithm 2	0.7	0.1	0.4	0.175	
Algorithm 3	0.02	1.0	0.501	0.0392	

Zekaşındaki modelde hangi algorithm kullanırsak data mantıklı olur sorusu sorulursa buna en iyi cevabı F1 score verir. Average üzerinden düşündürsek 3. algoritma da precision çok düşüktür ama recall çok yüksektir olayına rağmen en iyi average degerine sahip algoritmadır. Dolayısıyla average üzerinden değerlendirmeye yapmak olumlu değildir.

$$F_1 \text{ score} = \frac{1}{\frac{1}{2} \left(\frac{1}{P} + \frac{1}{R} \right)} = 2 \frac{PR}{P+R} \quad \boxed{\text{Harmonic mean of P and R}}$$

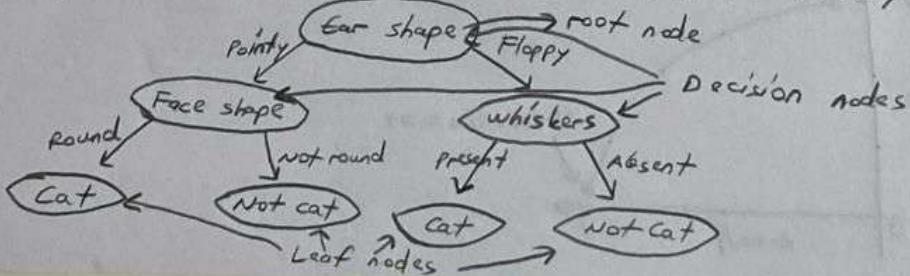
F1 score data düzük olan verileri data az önemseyen bir formüldür ve algoritmaların uyguladığımızda Algorithm 1 en iyi algorithm olarak görülmektedir.

* Harmonic mean is a way of taking an average that emphasizes the smaller values more.

* * Decision Tree

Cat classification Example			
Ear shape(x ₁)	Face shape(x ₂)	whiskers(x ₃)	cat
Pointy	Round	Present	1
Floppy	Not round	Present	1
Floppy	Round	Absent	0
Pointy	Not round	Present	0
Pointy	Round	Present	1
Pointy	Round	Absent	1
Floppy	Not round	Absent	0
Pointy	Round	Absent	1
Floppy	Round	Absent	0
Floppy	Round	Absent	0

Üstte 5 köpek 5 kedi örneğinin özelliklerini verdirdi ve binary classification probleminde



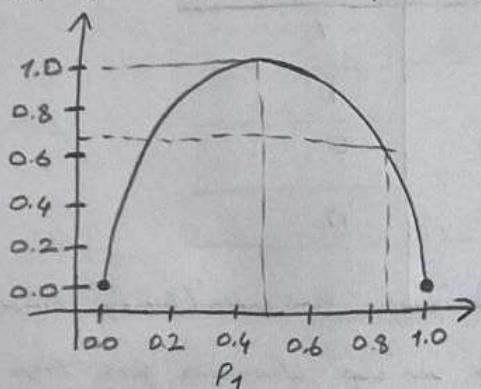
Çok fazla farklı türde decision tree modeli oluşturabilir. Bunların bazıları iyi performans verenlerdir. Bu sonucu verenlerin en iyisi decision tree algoritmasının asıl amacı olsaydı.

İyi şekilde genelliyen ve en iyi sonucu veren ağacı bulmakta biraz training set cross validation ve test set tarafından değerlendirilmelidir.

Decision tree oluştururken depth'in 3 den büyük olması ve olasılık en yüksek tree'nin tasarlanması önerilidir. Çünkü diğer durumlarda overfittinge yol açan kullanılsız bir tree olur. En iyi tree'nin bulunmasında da entropy hesabı yapılmıştır.

** Entropy as a measure of impurity

P_1 = fraction of examples that are cats



$$\begin{aligned} & \text{for 3 cat and 3 dog } p_1 = 3/6 \quad H(p_1) = 1 \\ & \text{for 5 cat and 1 dog } p_1 = 5/6 \quad H(p_1) = 0.65 \\ & \text{for 6 cat } p_1 = 6/6 \quad H(p_1) = 0.0 \\ & \quad (\text{impure}) \end{aligned}$$

Yukarıdaki veriler impure durumu içindir. Niçin ise pure bir durum oluşturmak istiyoruz?

$$P_0 = 1 - P_1$$

$$\begin{aligned} H(p_1) &= -p_1 \log_2(p_1) - p_0 \log_2(p_0) \\ &= -p_1 \log_2(p_1) - (1-p_1) \log_2(1-p_1) \end{aligned}$$

Note $\Rightarrow "0 \log 0" = 0$. $\log 0$ aslında tanımsız olmasına rağmen entropy hesabında $0 \log 0$ yapısını 0 olarak kabul ediyoruz.

The reduces of entropy = Information Gain choosing a Split

$$\begin{cases} P_1 = 5/10 = 0.5 \\ H(0.5) = 1 \end{cases}$$

Ear shape	Face shape	Whiskers
Pointy	Round	Present
Floppy	Not round	Absent
$P_1 = 4/5 = 0.8$	$P_1 = 4/7 = 0.57$	$P_1 = 3/4 = 0.75$
$H(0.8) = 0.72$	$H(0.57) = 0.99$	$H(0.75) = 0.81$
$H(0.5) - \left(\frac{5}{10} H(0.8) + \frac{5}{10} H(0.2) \right)$	$H(0.5) - \left(\frac{7}{10} H(0.57) + \frac{3}{10} H(0.33) \right)$	$H(0.5) - \left(\frac{4}{10} H(0.75) + \frac{6}{10} H(0.33) \right)$
$= 0.28$	$= 0.03$	$= 0.12$

↓
root node
Wigger than others
Information Gain

Decision Tree Learning

- Start with all examples at the root node.
- Calculate information gain for all possible features, and pick the one with the highest information gain.
- Split dataset according to selected feature, and create left and right branches of the tree.
- Keep repeating splitting process until stopping criteria is met:
 - when a node is 100% one class
 - when splitting a node will result in the tree exceeding a maximum depth
 - Information gain from additional splits is less than threshold
 - when number of examples in a node is below a threshold.

~~#~~ Features with three possible values:

ear shape(x ₁)	Face shape(x ₂)	whiskers(x ₃)	Cat(y)
Pointy	Round	Present	1
Oval	Not round	Present	1
Oval	Round	Absent	0
Floppy	Not round	Absent	0

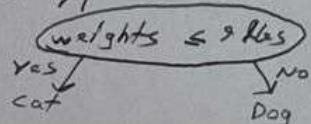
* One hot encoding: Bu yani bir feature'un 2den fazla türü varsa (binary durumu olmaz) bu türküm her birini feature olarak kabul edip var olup olmamasına göre veya o degeri verilenek olusturulmasından direkt istekli tablodaki 'ear shape' feature'u elde almak için yeniden düzenlenirse aşağıdaki gibi olur.

Ear shape	Pointy ears	Floppy ears	Oval ears	Face shape	whiskers	Cat
Pointy	1	0	0	Round	Present	1
Oval	0	0	1	Not round	Present	1
Oval	0	0	1	Round	Absent	0
Floppy	0	1	0	Not round	Absent	0

If a categorical feature can take on k values, create k binary features (0 or 1)

* Continuous values: var veya yok (1-0) gibi degerler yine sürekli degerler oluyor, mesela hayvanların ağırlığı gibi bu tarz durumda sürekli degeri x eksenine alıp y degeri olacak da cat veya not cat binary degerini grafixe döüp gördigimiz analitiklerde entropy hesabını yaparsak bize en iyi genelleştirilecek degeri verir.

en iyi



* Tek bir tree yapmak tıpkı degistikliklere karşı çok hassas bir yapıya sahip olmaktadır.
 Birden çok ağacı kullanmak "tree ensembles" data çözüm bir yapı sunar
 örneğin bir 3 feature'a sahipsek ve tek bir ağacımız varsa ve bu ağda sadece 2 feature
 kullanarak sonuca geldiysa bu sisteme tam olarak güvenemeyiz ve 3 feature'u da kullanarak
 farklı ağalar bulup teknik edilen sonuçları aralarında oylenerek data net sonuç verir.
 Bu yöntemleri kullanmak için çeşitli istatistiksel modeller vardır.
 ilk olarak nasıl farklı bir tree oluşturacağımızı disündirecekt. Bir torbaya
 4 farklı token atığımızı düşünelim sarı, kırmızı, yeşil ve mavi renkte olsun. Bu torbadan
 bir tane token seçip renkini kaydedip tekrar torbaya atınız sonra tekrar token çekiniz.
 Bu şekilde farklı tree'lerde alıcı edilebilir. Mesela sonuçlar söyle gelebilir:
 kırmızı, yeşil, sarı, mavi
 mavi, mavi, sarı, yeşil
 kırmızı, kırmızı, kırmızı, yeşil
 Bu torbadan 100 farklı 4 versişi olan ağacı yapmak ve her biri için sonuç bulmaya
 çalışıp en sonunda da sonuçları oylenip one göre karara varmak tek bir ağac yapabileceğinden
 çok daha iyi sonuçlar alıcı ederdir.

~~*** Random Forest Algorithm:~~

At each node, when choosing a feature to use to split, if n features are available, pick
 a random subset of $k \ln n$ features and allow the algorithm to only choose from
 that subset of features.

- if n is too large choose $k = \sqrt{n}$

~~**** XGBoost~~ algorithm bu konuda en sık tercih edilen ve en iyi sonuç veren algoritmidir.
 Given training set of size m

For $i=1$ to B :

use sampling with replacement to create a new training set of size m .
 But instead of picking from all examples with equal ($1/m$) probability, make it
 more likely to pick missclassified examples from previously trained trees
 Train a decision tree on the new dataset.

Bu algoritmin özelligi her tree yaplığında yanlış sınıflandırılan varlığı tespit etmek
 ve onları doğru sınıflandırma kadar diğer ağalarda da test etmekti (kombin bir matematiksel
 model'e sahiptir)

XGBoost bir insanın enstürmen galibiyetinin personen təmənin salmaya uyğunlaşdırılmasına əsasən işləməyi öyrəndi. Sonra birlikdən sonra digər personlər da etibarət öyrənməsinə ləğazdır.

XGBOOST (eXtreme Gradient Boosting)

- Open source implementation of boosted trees
- Fast efficient implementation
- Good choice of default splitting criteria and criteria for when to stop splitting
- Built in regularization to prevent overfitting
- Highly competitive algorithm for machine learning competitions (e.g. Kaggle competitions)

+ Using XGBoost:

Classification

```
from xgboost import XGBClassifier
model = XGBClassifier()
model.fit(X-train, y-train)
y-pred = model.predict(X-test)
```

Regression

```
from xgboost import XGBRegressor
model = XGBRegressor()
model.fit(X-train, y-train)
y-pred = model.predict(X-test)
```

Decision trees vs. Neural Networks

- Decision trees and tree ensembles (generally use XGBoost)
 - works well on tabular (structured) data.
 - Not recommended for unstructured data (images, audio, text)
 - Fast
 - Small decision trees may be human interpretable
- Neural Networks
 - works well on all types of data, including tabular (structured) and unstructured data.
 - May be slower than a decision tree
 - Works with transfer learning
 - when building a system of multiple models working together, it might be easier to string together multiple neural networks

Unsupervised learning, recommender systems and reinforcement learning

* Unsupervised Learning

- Clustering
- Anomaly detection

* Recommender systems

* Reinforcement learning

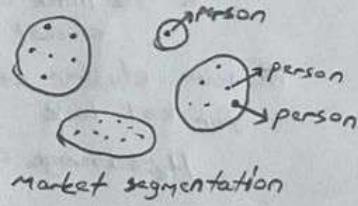
İlk olarak clustering nedir ve clustering algoritması hakkında gibi basit kisimlara bakınabiliz.

Supervised learning konusunu isterken elimizde (x, y) çiftleri oluyordu. Her x 'in y etiketi bu binary classification da olabilir dğeri de fakat şu an isteyeceğiniz unsupervised learningde çok önemlidir bir fark vardır. Unsupervised learning modellerinde sadece " x " değerleri vardır. x değerine karşılık gelen y değerleri yoktur ve biz onları öneleyelim (clustering) çalışırız.

- Applications of clustering

Grouping similar news,

- Growing skills
- Develop career
- stay updated with AI, understand how it affects your field of work.



DNA analysis, Astronomical data analysis, space exploration ve bunca benzer birçok farklı alanda clustering kullanılabılır.

** K-means clustering algorithm:

K-means algoritım temelinde şu şekilde işlenir. Bütün veriler düzlemede gösterilir ardından resfgele 2 farklı nokta seçiliğin biri kırmızı diğeri mavi olsun sonra her noktanın uzaklığını mavi ve kırmızı noktaya göre hesaplanır. Ardından hangi noktaya yakinsa onun rengini alır. Bütün noktalar renklendirildikten sonra kırmızı ve mavi renkli noktaların taneet iğin ayrı ayrı ortalarları bulunur (kirmiziların ortalaması, mavilerin ortalaması). Ardından ilk başta seçtiğiniz kırmızı ve mavi noktalar bu ortalara noktalara kaydırılmalı ve tüm süreç tekrar edilebilidir. En sonunda iki farklı kime genetir.

Randomly initialize k cluster centroids $\mu_1, \mu_2, \dots, \mu_k$

repeat { # Assign points to cluster centroids

for $i=1$ to m

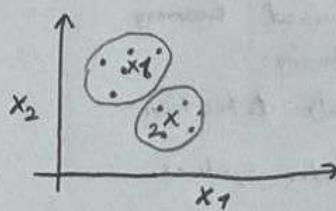
$c^{(i)} :=$ index (from 1 to k) of cluster centroid closest to $x^{(i)}$

Move cluster centroids

for $k=1$ to k

$\mu_k :=$ average (mean) of points assigned to cluster k }

$$\mu_1 = \frac{1}{4} [x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}]$$



* Cost functions for k -means

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

Repeat { # Assign points to cluster centroids

for $i=1$ to m

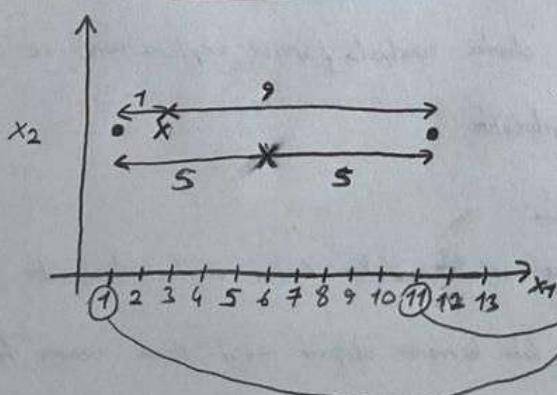
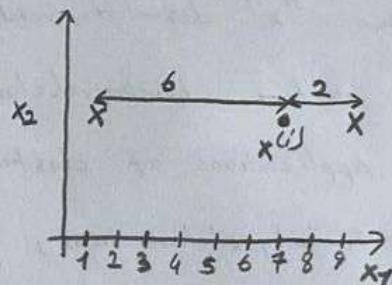
$c^{(i)} :=$ index of cluster centroid closest to $x^{(i)}$

Move cluster centroids

for $k=1$ to k

$\mu_k :=$ average of points in cluster k }

Moving the centroid



$$\frac{1}{2}(1^2 + 9^2) = 41$$

$$\frac{1}{2}(1+11) = 6$$

$$\frac{1}{2}(5^2 + 5^2) = 25$$

25 is much smaller average square distance than 41.

Random Initialization:

for $i=1$ to 100 {

50-1000

Randomly initialize k -means

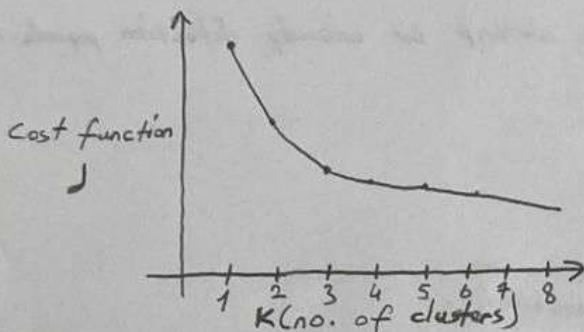
Run k -means. Get $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k$

Computer cost function (distortion)

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k)$$

} Pick set of clusters that gave lowest J

k degeri yani kag tone cluster olmasi gerektigi bir problemdir ve bunun hakkında bazi fikirlerimiz vardir



Örnek

Soldakı grafik gibi bir grafik elde edersek k degerini 3 olarak seceriz çünkü 3'e kadar çok hızlı bir atılıf var 3den sonra ise ciddi bir yavaşlma. Bu grafik insan düşüncesi benzetilebilir ve bu da "elbow method" denir.

Eğer üstteki grafik gibi bir şekilde sahip değilsek daha stabil gözükse diğer anodarların sağlıdığı performansa bakınız.

→ often, you want to get clusters for some later (downstream) purpose.

Evaluate k -means based on how well it performs on that later purpose.

Anomaly Detection

Anomaly detection algorithm, unlabeled veritabındaki abnormal durumları tespit etmeye yararlı. Aircraft engine üzerinden düşündürsek bir aircraft engine'in olması yada kötü bir durumla başı karsıya kalması en son istenek durumudur.

Aircraft engine features:

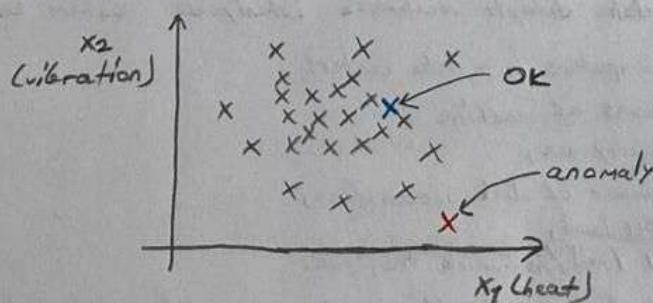
x_1 = heat generated

x_2 = vibration intensity

⋮

$$\text{Dataset} = \{x^{(1)}, x^{(2)}, \dots, x^{(n)}\}$$

New engine = x_{test}



Elmizdeki verileri featurelara göre düzleme aktardıktan sonra test degerinin normal veya anomali şeklinde gözlemlayıp sınıflandırabilirim. Fakat bu sınıflandırma doğru mu veya anomaly olan kisim problem yaratır mi gibi sorular gelebilir.

Density Estimation:

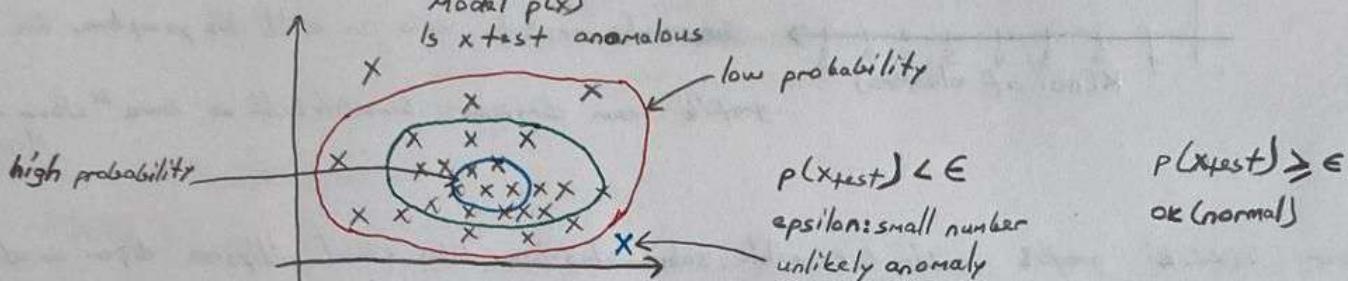
$$\text{Dataset} = \{x^{(1)}, x^{(2)}, \dots, x^{(M)}\}$$

Elinizde bir veriseti var ve bunu döktürme aktarır. Bir anomaly detection yapmak istiyorsanız neyi bilgisiyorsanız bu problemi nasıl çözer?

probability of x being seen in dataset

$$\text{Model } p(x)$$

Is x_{test} anomalous



Anomaly Detection Example:

Fraud detection:

- $x^{(i)}$ = features of user i 's activities
- Model $p(x)$ from data
- Identify unusual users by checking which have $p(x) < \epsilon$

Bir nekterin verisi su taraflı soruların cevaplarından oluşturulur:

how often log in?

how many web pages visited?

transactions?

posts? typing speed?

Eğer edilen modele göre şifeli bir durumlar tespit edilir ve extra uygulamalarda gerekleme olursa, yoksa yanlış alarm mı不断增强. Dolayısıyla sunu söyleyebiliriz: olgandığı veya normal bir durum tespiti için şirketler tarafından su taraflı uygulamalar kullanılır.

Bu taraflı normal durumlar data datayı inclemek isteniyorsa sistem kontrol edilebilir.

Monitoring computers in a data center:

$x^{(i)}$ = features of machine i

- x_1 = memory usage
- x_2 = number of disk accesses/sec
- x_3 = CPU load
- x_4 = CPU load/network traffic

Bu features değerlendirilerek hacketme durumu veya sisteme/veya ağ

talansız herhangi bir problem var mı diye değerlendirilir.

(31)

Bir örnek' sayfada' anomaly detection için probability of X değişti. Bunu işin kullanıldığı
olasılık metodu "Gaussian (normal) distribution" kullanılır

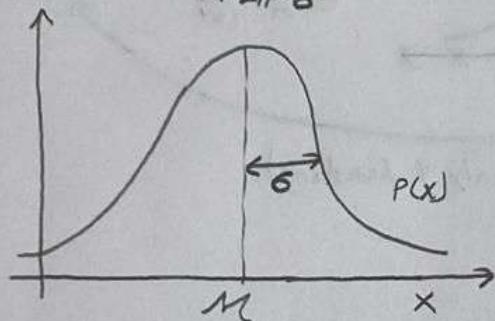
Gaussian (Normal) distribution:

Say x is a number.

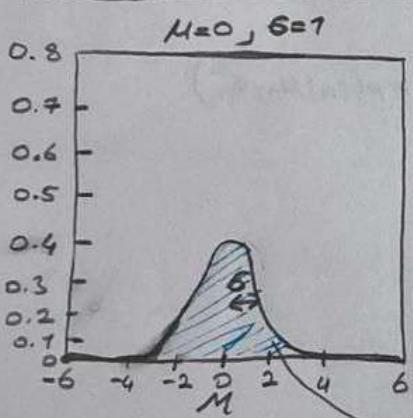
standard dev. σ

Probability of x is determined by a Gaussian with mean μ , variance σ^2 .

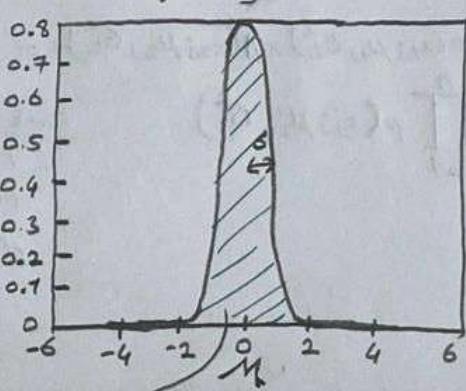
$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad \pi=3.14$$



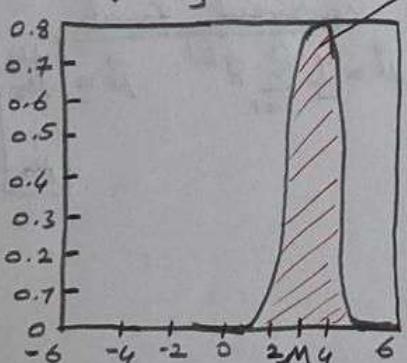
Gaussian distribution example:



$\mu=0, \sigma=0.5$



$\mu=3, \sigma=0.5$

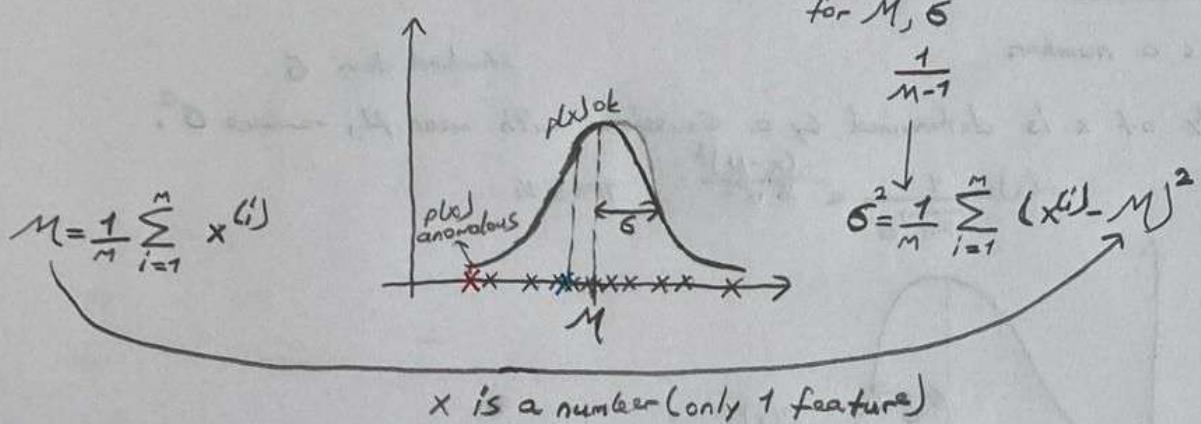


bu alanlar 1'e eşittir

Bunlar μ ve σ degerine göre degisen
orneklendir

Parameter estimation:Dataset: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

maximum likelihood

for μ, σ^2 Density estimation:Training set = $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ Each example $x^{(i)}$ has n features

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$p(\vec{x}) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2) \times p(x_3; \mu_3, \sigma_3^2) \times \dots \times p(x_n; \mu_n, \sigma_n^2)$$

$$= \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)$$

Given

$$p(x_1 = \text{high temp}) = \frac{1}{10}$$

$$p(x_2 = \text{high vib}) = \frac{1}{20}$$

$$p(x_1, x_2) = p(x_1) \times p(x_2)$$

$$= \frac{1}{10} \times \frac{1}{20} = \frac{1}{200}$$

Anomaly Detection Algorithms:1. Choose n features x_i that you think might be indicative of anomalous examples.2. Fit parameters $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

vectorized formula

$$\vec{\mu} = \frac{1}{m} \sum_{i=1}^m \vec{x}^{(i)}$$

$$\vec{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix}$$

3. Given new example x , compute $p(x)$:

$$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if $p(x) < \epsilon$

Anomaly Detection

vs.

Supervised Learning

- 1) Very small number of positive examples ($y=1$). (0-20 is common)
- 2) Large number of negatively = 0 examples
- 3) Many different "types" of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.

FraudDiger örnekler

- * Manufacturing - finding known previously seen defects
- * Weather prediction (sunny/rain/etc.)
- * Diseases classification

- 1) Large number of positive and negative examples.
- 2) Enough positive examples for algorithm to get a sense of what positive examples are like; future positive examples likely to be similar to ones in training set.

email **spam**Diger örnekler

- * Manufacturing - finding new previously unseen defects in manufacturing. (e.g. aircraft engines)
- * monitoring machines in a data center

→ Hangi feature'ı segmeliyiz sorusuna cevap ararsak

Data enekti sayfalarda çizilen gaussian distribution examples ekrandaki grafiklere benzeyen grafik gibi bir sekil olursa. Eğer öbeklendirme farklýysa ve istedigimiz gibi bir grafik göründiyorsa zorlana grafikler elde edilebilir.

$$\begin{aligned}x_1 &\leftarrow \log(x_1) \\x_2 &\leftarrow \log(x_2+1) \\x_3 &\leftarrow \sqrt{x_3} \\x_4 &\leftarrow x_4^{1/3}\end{aligned}$$

** in Python, plt.hist(np.log(x), bins=50);

asthetik gibi yazmak hata yapar. ufat bir trick kullanip plt.hist(np.log(x+0.01), bins=50) yatsarak aktivitesizlikten kurtulmus olur.

** Anomaly detection problemlerde data kesin sonuçlar elde etmek istiyorsak tek feature üzerinden test yerine 2 feature veya data points düzlene abartıldıkları sonra test edilebilir.

** monitoring computers in a data center denginde ise yeni feature'lar oluşturulabilir.

$$x_5 = \frac{\text{CPU load}}{\text{network traffic}}$$

$$x_6 = \frac{(\text{CPU load})^2}{\text{network traffic}}$$

gibi

Recommender Systems:

Bu konu Amazon gibi alışveriş sitelerinden netflix gibi dijital içerik platformlarına kadar
dan ve bu kişiye özel içerik önermenin nasıl yapıldığını ele alır.

Predicting movie ratings

User rates movies using zero to five stars

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)	
Love at last	5	5	0	0	$n_u = \text{number of users}$
Romance forever	5	?	?	0	$n_m = \text{number of movies}$
Cute puppies of love	?	4	0	?	$r(i,j) = 1$ if user j has rated movie i
Nonstop car chases	0	0	5	4	$y^{(i,j)} = \text{rating given by user } j$ to movie i
Swords vs. karate	0	0	5	?	(defined only if $r(i,j)=1$)

$n_u = 4$ $r(1,1) = 1$
 $n_m = 5$ $r(3,1) = 0$ $y^{(3,2)} = 4$

Bu tarz bir yorumın amacı sudur. Kisiinin data önce puan vermediği (?) verilere bakıp
hangisini bu kişiye öncelikle kisi' öndeki 5 puanla ya da olabilirce en yüksek degerle puanları.

→ What if we have features of the movies?

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)	x_1 (romance)	x_2 (action)	$n_u = 4$
Love at last	5	5	0	0	0.9	0	$n_m = 5$
Romance forever	5	?	?	0	1.0	0.01	$n = 2$ (x_1, x_2)
Cute puppies of love	?	4	0	?	0.99	0	$x^{(1)} = \begin{bmatrix} 0.9 \\ 0 \end{bmatrix}$
Nonstop car chases	0	0	5	4	0.1	1.0	$x^{(3)} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix}$
Swords vs. karate	0	0	5	?	0	0.9	

For user 1: Predict rating for movie i as: $w^{(1)} \cdot x^{(i)} + b^{(1)}$ ← just linear regression

$$w^{(1)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix} \quad b^{(1)} = 0 \quad x^{(3)} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix} \quad w^{(1)} \cdot x^{(3)} + b^{(1)} = 4.95 \quad \checkmark$$

→ For user j : predict user j 's rating for movie i as $w^{(i)} \cdot x^{(j)} + b^{(i)}$

Bu formül her gün üye için uygulanabilir ve elinizdeki feature'lara göre
kullanıcının ne puan vereceği tahmin edilebilir.

Cost Function

Notation:

$r^{(i,j)} = 1$ if user j has rated movie i (0 otherwise)

$y^{(i,j)}$ = rating given by user j on movie i (if defined)

$w^{(i,j)}, b^{(i)}$ = parameters for user j

$x^{(i)}$ = feature vector for movie i

For user j and movie i , predict rating: $w^{(i)} \cdot x^{(i)} + b^{(i)}$

$m^{(i)}$ = no. of movies rated by user j

To learn $w^{(i)}, b^{(i)}$

$$\min_{w^{(i)}, b^{(i)}} J(w^{(i)}, b^{(i)}) = \frac{1}{2} \sum_{\substack{i: r^{(i,j)}=1 \\ j=1}} (w^{(i)} \cdot x^{(i)} + b^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^n (w_k^{(i)})^2$$

number of features
sadece puanlanan filmler

Bu formül klasik mean squared error formülündür amaç w ve b değerlerini en uygun değer seçerek hatalı minimum durumda getirmektedir. Tahmin J ile asıl değer arasındaki farklar

To learn parameters $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots, w^{(n)}, b^{(n)}$ for all users:

$$J(w^{(1)}, b^{(1)}, \dots, w^{(n)}, b^{(n)}) = \frac{1}{2} \sum_{j=1}^m \sum_{\substack{i: r^{(i,j)}=1 \\ i=1}} \frac{(w^{(i)} \cdot x^{(i)} + b^{(i)} - y^{(i,j)})^2}{f(x)} + \frac{\lambda}{2} \sum_{j=1}^m \sum_{k=1}^n (w_k^{(i)})^2$$

Collaborative Filtering:

cost function to learn $w^{(1)}, b^{(1)}, \dots, w^{(n)}, b^{(n)}$:

$$\min_{w^{(1)}, b^{(1)}, \dots, w^{(n)}, b^{(n)}} \frac{1}{2} \sum_{j=1}^m \sum_{\substack{i: r^{(i,j)}=1 \\ i=1}} (w^{(i)} \cdot x^{(i)} + b^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^m \sum_{k=1}^n (w_k^{(i)})^2$$

cost function to learn $x^{(1)}, \dots, x^{(m)}$:

$$\min_{x^{(1)}, \dots, x^{(m)}} \frac{1}{2} \sum_{i=1}^m \sum_{\substack{j: r^{(i,j)}=1 \\ j=1}} (w^{(i)} \cdot x^{(i)} + b^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^m \sum_{k=1}^n (x_k^{(i)})^2$$

Put them together:

$$J(w, b, x) = \frac{1}{2} \sum_{j=1}^m \sum_{\substack{i: r^{(i,j)}=1 \\ i=1}} (w^{(i)} \cdot x^{(i)} + b^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^m \sum_{k=1}^n (w_k^{(i)})^2 + \frac{\lambda}{2} \sum_{i=1}^m \sum_{k=1}^n (x_k^{(i)})^2$$

Gradient Descent (collaborative filtering)

Linear regression (course 1)

repeat {

$$w_i^{(ij)} = w_i^{(ij)} - \alpha \frac{\partial}{\partial w_i^{(ij)}} J(w, b, x)$$

$$b^{(ij)} = b^{(ij)} - \alpha \frac{\partial}{\partial b^{(ij)}} J(w, b, x)$$

$$x_k^{(ij)} = x_k^{(ij)} - \alpha \frac{\partial}{\partial x_k^{(ij)}} J(w, b, x)$$

}

Binary labels: favs, likes and clicks:

True 0 değerleri yani binary labeller ne zaman kullanılır, önem kazanır bunca değerden birini.

Example Applications:

1. Did user j purchase an item after being shown? 1, 0, ?
2. Did user j fav/like an item? 1, 0, ?
3. Did user j spend at least 30 sec with an item? 1, 0, ?
4. Did user j click on an item? 1, 0, ?

* Meanings of ratings:

1 - engaged after being shown item

0 - did not engage after being shown item

? - item not yet shown

From regression to binary classification:

Previously:

Predict $y^{(ij)}$ as $w^{(ij)} \cdot x^{(ij)} + b^{(ij)}$

For binary labels:

Predict that the probability of $y^{(ij)} = 1$
is given by $g(w^{(ij)} \cdot x^{(ij)} + b^{(ij)})$

where $g(z) = \frac{1}{1+e^{-z}}$

cost function for binary application:

$$\text{Loss for binary labels } y: f_{w, b, x}(x) = g(w \cdot x + b) - y$$

$$L(f_{w, b, x}(x), y) = (g(w \cdot x + b) - y) \log(g(w \cdot x + b)) + (1 - g(w \cdot x + b)) \log(1 - g(w \cdot x + b)) \quad \leftarrow \text{Loss for single example}$$

$$\sum_{i=1}^n L(f_{w, b, x}(x_i), y_i)$$

Mean NormalizationUsers who have not rated any movies

Movie	Alice(1)	Bobi(2)	Carrie(3)	Dave(4)	Eve(5)
Love at last	5	5	0	0	?
Romance forever	5	?	?	0	?
Cute puppies after	?	4	0	?	?
Nonstop car chases	0	0	5	4	?
Swords vs. Karate	0	0	5	?	?

Yukarıda bazı kisiyein ratinglerini verilmisti. Bu ratingleri kullanarak Eve'nin tas puan verecagine dair mantiki, bir sistemi yapalim. öncelikle tüm degerleri bir matrise dökelim.

$$\begin{bmatrix} 5 & 5 & 0 & 0 & 2 \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \xrightarrow{\text{Average}} \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}$$

ortalama değer
matrisi

$$\begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & 3 & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

ilk matrisden ortalama değer matrisini
ekardiktan sonraFor user j , on movie i predict:

$$w^{(ij)} \cdot x^{(ij)} + b^{(ij)} + \mu_i \quad (\text{ortalama degeri ekardigimiz icin burda tekrar ekliyoruz})$$

UserS(Eve):

$$w^{(5)} = [8] \quad b^{(5)} = 0 \quad w^{(5)} \cdot x^{(5)} + b^{(5)} + \mu_5 = 2.5$$

* Tensorflow implementation of collaborative filtering:Derivatives in ML

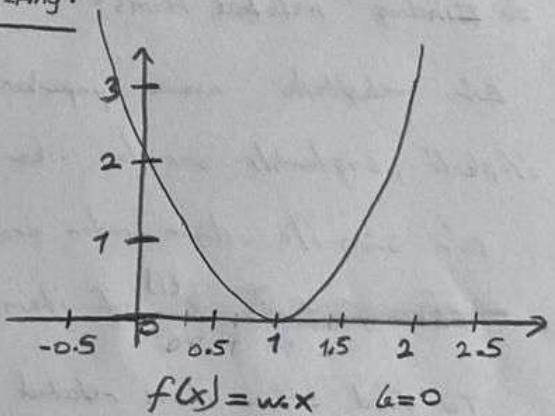
Gradient descent algorithm

Repeat until convergence

$$w = w - \alpha \frac{\partial J(w, b)}{\partial w}$$

Learning rate
Derivative

$$b = b - \alpha \frac{\partial J(w, b)}{\partial b} \leftarrow b = 0$$



Custom Training Loop:

$w = tf.Variable(3.0)$ → $Tf.$ variables are the parameters we want to optimize

$x = 1.0$

$y = 1.0$

$alpha = 0.01$

$iterations = 30$

Auto Diff

Auto Grad

for iter in range(iterations):

with $tf.GradientTape$ as tape:

$$fwk = w * x \leftarrow f(x)$$

$$cost_J = (fwk - y)^2$$

$$[dJ/dw] = tape.gradient(cost_J, [w])$$

$$w.assign_add(-alpha * dJ/dw)$$

↑
tf.variables require special function to modify

* Implementation in Tensorflow with Adam optimizer:

Gradient descent algorithm

Repeat until convergence

$$w = w - \alpha \frac{\partial}{\partial w} J(w, b, x)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w, b, x)$$

$$x = x - \alpha \frac{\partial}{\partial x} J(w, b, x)$$

optimizer = keras.optimizers.Adam(learning_rate=1e-1)

iterations = 200

for iter in range(iterations):

with $tf.GradientTape()$ as tape:

cost_value = config.CastFuncV(X, w, b, rnorm, R,
num_users, num_movies, lambda)

grads = tape.gradient(cost_value, [X, w, b])

optimizer.apply_gradients(zip(grads, [X, w, b]))

* Automatic differentiation

* Collaborative filtering

* Finding related items:

Bir web sitesinde arama yaparken, rada bir film sitesinde film sepetlerin sonra genelde ilişkili, bağlantıları sonular ekler ve önerilerdeki bu nasıl oluyor? (collaborative algorithm ile)

Bir ürün ile diğer ürünler arasındaki benzerlik ve farklılık anımlı olan kisimlar.

The features $x^{(i)}$ of item i are quite hard to interpret.

To find other items related to it, find item k with $x^{(k)}$ similar to $x^{(i)}$

i.e. with smallest
distance

$$\sum_{l=1}^n (x_l^{(k)} - x_l^{(i)})^2$$

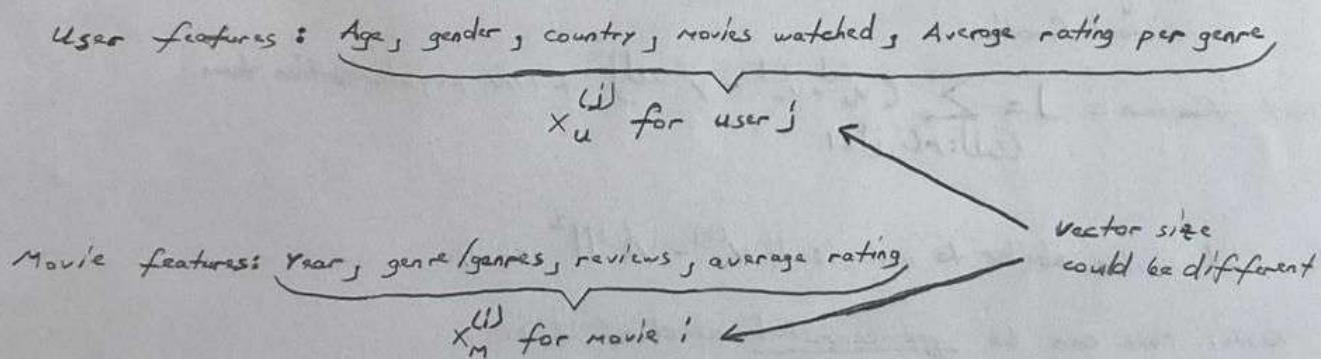
Limitations of Collaborative Filtering

- Cold start problem. How to
 - rank new items that few users have rated?
 - show something reasonable to new users who have rated few items?
- Use side information about items or users:
 - Items: genre, movie stars, studio,
 - Users: Demographics (age, gender, location), expressed preferences,

Collaborative filtering vs Content-based filtering:

- Collaborative filtering: Recommend items to you based on ratings of users who gave similar ratings as you.
- Content-based filtering: Recommend items to you based on features of user and item to find good match

Examples of user and item features:



Content-based filtering: Learning to match

Predict rating of user j on movie i as

$$w^{(i)} \cdot x_u^{(i)} + b^{(i)}$$

$$v_u^{(i)} \cdot v_m^{(i)}$$

computed from $x_u^{(i)}$

computed from $x_m^{(i)}$

users' preferences $v_u^{(i)} = \begin{bmatrix} 4.9 \\ 0.1 \\ \vdots \\ 3.0 \end{bmatrix}$ likes

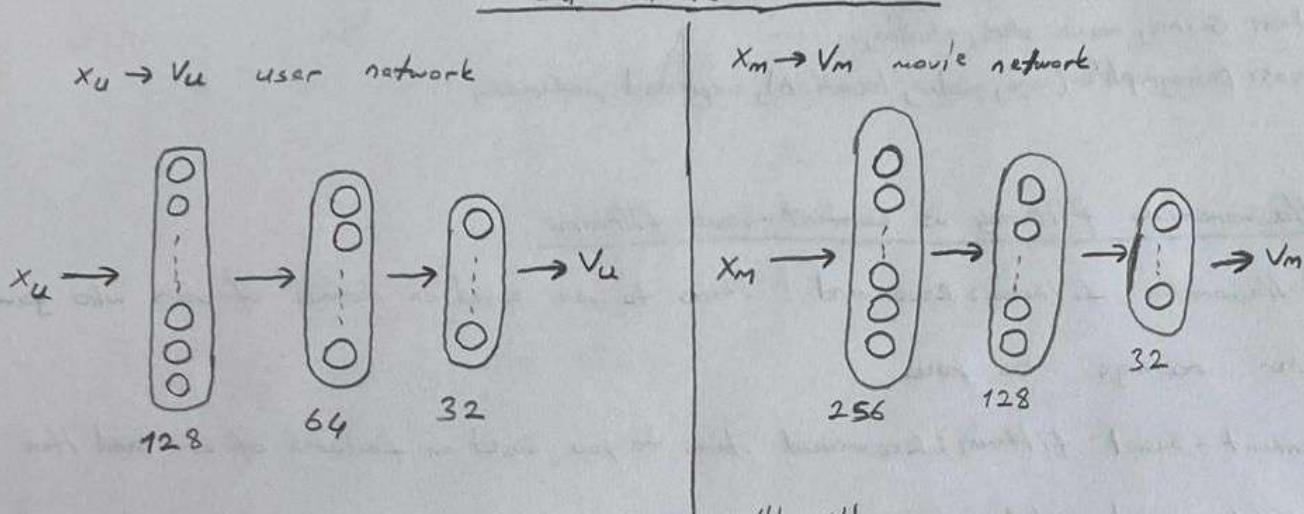
movie features $v_m^{(i)} = \begin{bmatrix} 4.5 \\ 0.2 \\ \vdots \\ 3.5 \end{bmatrix}$ romance action

(40)

Content-based filtering'in en iyi özelligi neural network ile ilişkilendirilip bir sonuca varılabilir. Çok sayıda nöral ağlar ile olan fırı vardır.

→ Deep learning for content-based filtering:

Neural Network Architecture



$$\text{Prediction} = v_u^{(ij)} \cdot v_m^{(ij)}$$

$g(v_u^{(ij)} \cdot v_m^{(ij)})$ to predict the probability that $y^{(ij)}$ is 1
sigmoid function

$$\text{cost function} = J = \sum_{(i,j) : r(i,j)=1} (v_u^{(ij)} \cdot v_m^{(ij)} - y^{(ij)})^2 + \text{NN regularization term}$$

To find movies similar to movie i : $\| v_m^{(i)} - v_m^{(j)} \|$ small

Note: This can be pre-computed ahead of time.

Bu özellik sayesinde kullanici baska zaman tekrar siteye geldiginde en yakini, en benzer 10-20 icerde ile karşılaşırs.

**** Nöral ağları kullanmanın faydalardan biri bir çok nöral ağ yapısını oluşturarak ve daha hızlı sistemlerde kullanılmamasını sağladığını. Üstteki esim bunun bir örneğidir.

→ Tensorflow implementation of content-based filtering:

```
user_NN = tf.keras.models.Sequential([
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(32) ])
```

```
item_NN = tf.keras.models.Sequential([
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(32) ])
```

(41)

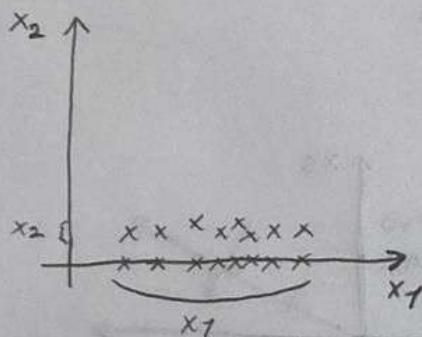
```

input_user = tf.keras.layers.Input(shape=(num_user_features))
vu = user_NN(input_user)
vu = tf.linalg.l2_normalize(vu, axis=1)
input_item = tf.keras.layers.Input(shape=(num_item_features))
vm = item_NN(input_item)
vm = tf.linalg.l2_normalize(vm, axis=1)
output = tf.keras.layers.Dot(axes=1)([vu, vm])
model = Model([input_user, input_item], output)
cost_fn = tf.keras.losses.MeanSquaredError()

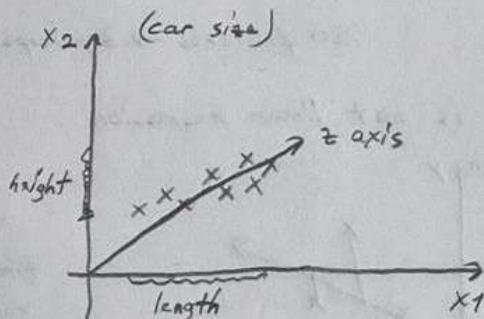
```

Principal Component Analysis:

Bu aşamada asıl tanealtı olan olay yüksek boyutlu ya da yüksek değişkenli olayları daha düşük düzleme indirmektedir. Mesela 3D lir problemleri 2D'ye indirebilir gibi söyleyebilir.



we can just take to reduce number of features

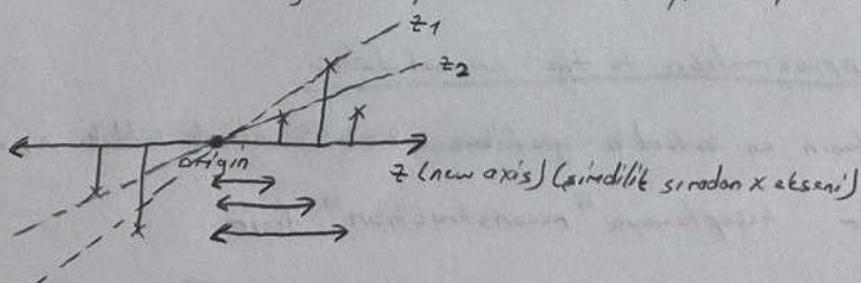


PCA: find new axis and coordinates use fewer numbers to capture "size" feature.

Genelde 3D düzleme aktarılmış verikerde eksenler etrafında oynanırsa dat lir sızı gibi bir bakış açısı elde edilebilir ve 3D veri 2D'ye indirebilir.

→ PCA Algorithm:

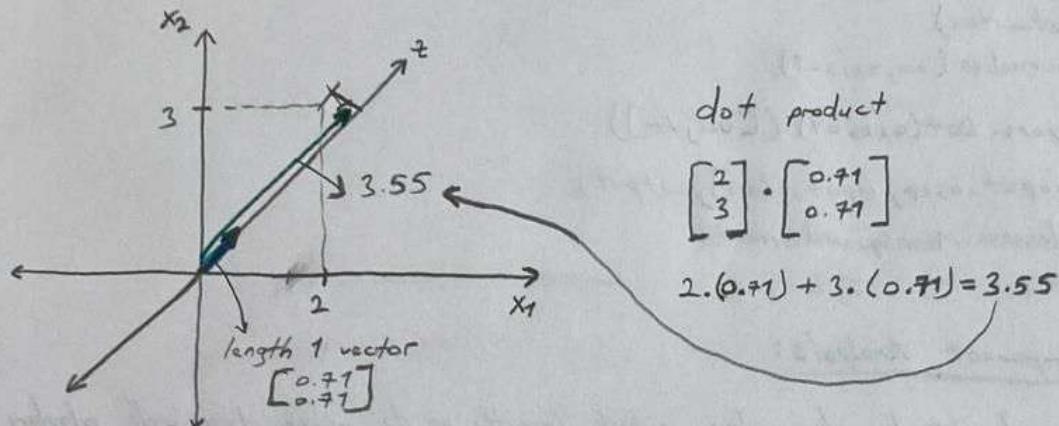
Üstteki koordinat sistemlerindeki gibi x ve y veya x₁ ve x₂ düzlemlerine sahip olmak istenip orası yani bir düzleme sahip olanın datalara göre en iyi düzleme ulaşmayı deneyebiliriz.



z_1 ve z_2 örnek düzlemlerinde olduğu gibi eksen iki dirdikte data iji lir sonucu ulaşırız. Ayrıca oluşan bu eksenin "principal component" denir.

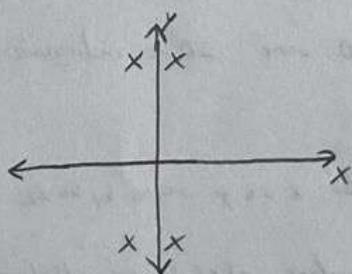
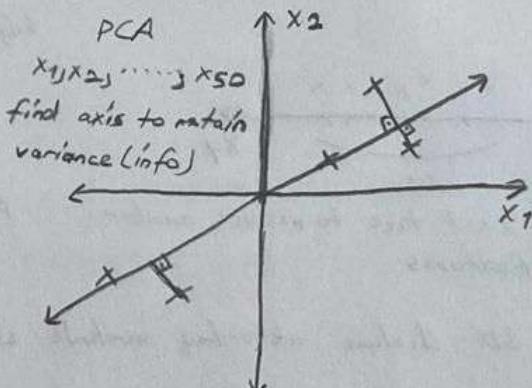
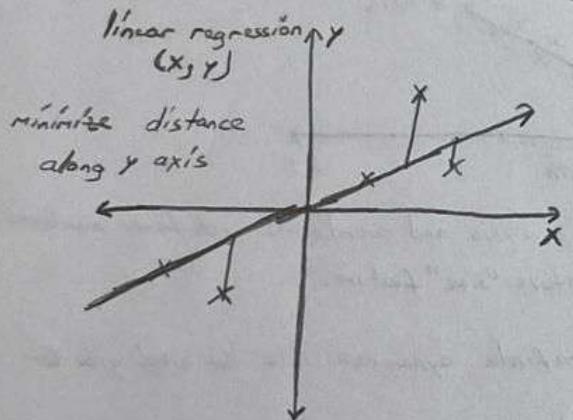
(42)

PCA projectionda kiti bir eksen segersek verilenin birlesine yaklastigini ve ayrim yapmanin bilgi edinmenin tar oldugunu gorunt. Fakat iyi bir eksen segersek veriler segitlenen ve birbirinden uzaklasir dolayisyla bilgi edinmek kolaylasir. Sci-kit learn modulu principal component otomatik bulunur.



** Eger birden fazla eksen olusturmak istiyorsak eksenler birlesine dik olmalıdır (90°)
50 features \rightarrow 3 principal component

***** PCA is not linear regression



Eger soldaki gibi bir verisine sahipse
linear regression x esenini sagar PCA ise y
esenini sagar

→ Approximation to the original data:

Sayfanin en ustindeki grafikte $z=3.55$ olarak verdilen x_1 ve x_2 degerini yaklasik olarak tekrar hesaplamaya "reconstruction" denir.

$$3.55 \times \begin{bmatrix} 0.71 \\ 0.71 \end{bmatrix} = \begin{bmatrix} 2.52 \\ 2.52 \end{bmatrix} \text{ close to } (2, 3)$$

(43)

PCA in Sci-kit learn

optional pre-processing: Perform feature scaling (verileri karsılıklı olarak aralıklara getirme)

1. "fit" the data to obtain 2 (or 3) new axes (principal components)

fit includes mean normalization

2. Optionally examine how much variance is explained by each principal component.

3. Transform (project) the data onto the new axes.

Example

```
X = np.array([[1, 1], [2, 1], [3, 2],
              [-1, -1], [-2, -1], [-3, -2]])
```

```
pca_2 = PCA(n_components=1)
```

pca_2.fit(X) *eksen sayısı*

pca_2.explained_variance_ratio_

```
X_trans_2 = pca.transform(X)
```

```
X_reduced_2 = pca.inverse_transform(X_trans_2)
```

Reinforcement Learning

Ticarette, günlük hayatta çok sık kullanılan bir türdür.

Reinforcement Learning nedir? Bu soruya şu şekilde cevap verebiliriz Bir drone ya da helikopter usunmak için joystick'e ihtiyac duyabilir ve uzaktan kumandalı araba sürer gibi (buna senzör) havada dengede tutarak hareket etmemeyi deneyebiliriz. Peki bunu biz de computer yapsaydı nasıl olurdu iste bu noktada Reinforcement Learning konuya dahil oluyor.

position of helicopter → how to move control sticks

state s → action a

(position, orientation
speed information)

(push the two control sticks to keep helicopter balanced in the air and flying and without crashing)

x → y

Neural network ile sistemi eğittiğimizi düşünelim. Bunun karşısında en iyi y'yi bulmak isteriz. Fakat bu diğer konulara göre karışık bir durumdur. Az sola ya da çok sola git veya yükseliş ya da alçal gibi durumlardan hangisi seviyor? Bu yüzden supervised learning yaklaşımı got iyi çalışmaz.

Bu noktada sıklıkla bilinen reward function bulanılabilir. Sisteme kendi aksiyon alması, için için vermek ve aldığı aksiyona göre positive ya da negative geri bildirim vermek gibi düşünelbilir. (kopek eğitimi gibi)

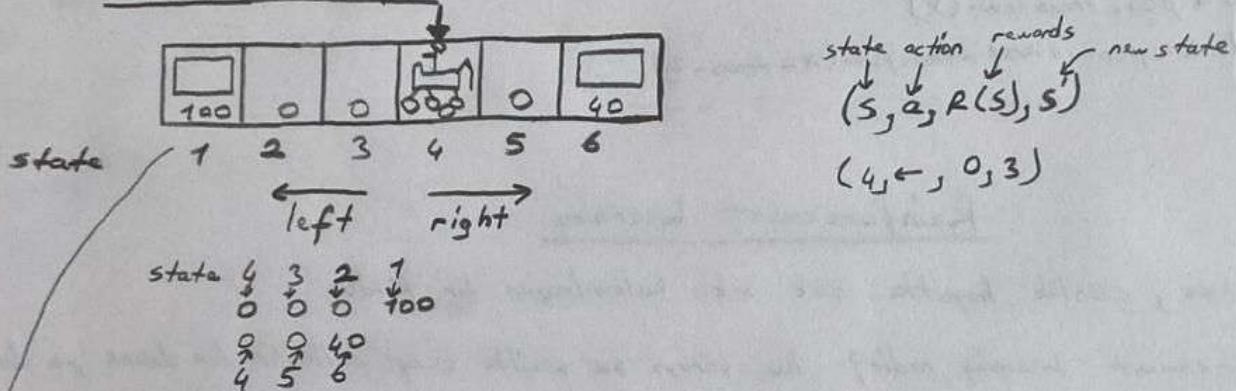
positive reward: helicopter flying well (+1)

negative reward: helicopter flying poorly (-1000)

Robot dog bir ömet olasılık engelleri aşılırsa puan kazanır aksamadıkça kaybeder.

- Applications:
- Controlling robots
 - Factory optimization
 - Financial (stocks) trading
 - Playing games (including video games)

Mars Rover Example:



The Return in reinforcement learning:

Return durumu şu şekilde değerlendirilebilir. Yukarıda verdigimiz örnekte robot 2 adım atıp 40 puanı mı alırsa yoksa 3 adım atıp 100 puan mı alırsa gibi bu nedenle ömeklerde en uygun karar nedir onun incelenmesidir.

$$\text{Return} = 0 + (0.9)0 + (0.9)^20 + (0.9)^3 \cdot 100 = 72.9$$

$$\text{Return} = R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$$

Discount factor is generally close to 1 but lower than 1 like 0.9, 0.99

Example of Return

for $\gamma = 0.5$

100	50	25	12.5	6.25	40
100	0	0	0	0	40
1	2	3	4	5	6

The return depends on the actions you take

100	2.5	5	10	20	40
100	0	0	0	0	40
1	2	3	4	5	6

100	50	25	12.5	20	40
100	0	0	0	0	40
1	2	3	4	5	6

Return kisminin olayı, işe, olaya göre değişebilir. Eğer finansal uygulama için düşünürsek takisitler ya da borcu ileri zamana aktarmak maliyetli olabilir. Şimdi geleceğin parası bugünden degerinden az olacaktır.

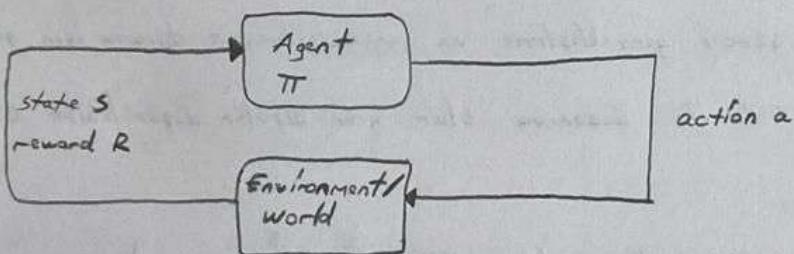
Policy	1	2	3	4	5	6
100	←	←	→	→	40	
state	s	π	a	$\pi(s)$	$\pi(a)$	$\pi(s,a)$
100	←	←	←	←	40	
100	→	→	→	→	40	
100	←	←	→	→	40	

$\pi(s) = a$
 $\pi(2) = \leftarrow$
 $\pi(3) = \leftarrow$
 $\pi(4) = \leftarrow$
 $\pi(5) = \rightarrow$

A policy is a function $\pi(s)=a$ mapping from states to actions, that tells you what action a to take in a given state s .

* * Markov Decision Process (MDP):

MDP aslında şu ana kadar yaptığımız her seyin özetini geldi. MDP şu anlit durumu önemser. Bu duruma nasıl geldik ya da bundan önceki herhangi bir öneni yoktur.



State action value function (Q-function):

$Q(s,a) = \text{Return if you}$
 \downarrow
 $\begin{array}{l} \bullet \text{ start in state } s. \\ \bullet \text{ take action } a \text{ (once).} \\ \bullet \text{ then behave optimally after that.} \end{array}$

100	100	50	12.5	25	62.5	12.5	10	6.25	20	40	40
100	0	0	0	0	0	0	0	40	40		

1 2 3 4 5 6

Her kutucunun solundaki değer sola gitmenin faydası sağindaki ise sağa gitmenin faydasıdır.

Hangi değer daha büyükse o yöne gitmek daha mantıklıdır.

* * Bellman Equations:

$Q(s,a) = \text{Return if you}$
 $\begin{array}{l} \bullet \text{ start in state } s. \\ \bullet \text{ take action } a \text{ (once).} \\ \bullet \text{ then behave optimally after that.} \end{array}$

s : current state

$R(s)$ = reward of current state

a : current action

s' : state you get to after taking action a

a' : action that you take in state s'

$$Q(s,a) = R(s) + \gamma \max_{a'} Q(s',a')$$

Bellman Equation'in kısaca olayı şudur

$$Q(s, a) = R_1 + \gamma R_2 + \gamma^2 R_3 + \dots \\ = R_1 + \gamma(R_2 + \gamma R_3 + \dots)$$

ilk denklemi 2. haline getirince ise ise olgunuz bellman equation'lar görünür. Dolayısıyla daha büyük bir problemi dinamik teknikde çözütmek için düşünülebilir.

Random (stochastic) Environment:

Bu konu aslında bulunulan çevre ile ilgili'dir. Daha öncesinde konuştugumuz konu tameren teorik olarak her şey normal iken gerçekleşecekti olsaydı. Fakat robotun bulunduğu çevrin tayfı ise veya asırı nüzgarın olduğu bir ortam var ise röva buna neden çevre şartlarından dolayı robot istenilen görevi yerine getiremeyebilir (örnek olarak state 3'deki bir robota 2 konumuna git konutu verildikten sonra çevre etkisi yaradınca state 4'a gitmesi gibi düşünülebilir). Bunden dolayı istenilen görevi gerçekleştirmeye ve gerçekleştirmeye durumu için yıldızlar atmalıdır. 0.9 ile 0.85, 0.7 ile 0.65 olur gibi değerler düşünülebilir. Bunun igeri bir istatistik yapısı kullanırız.

$$\text{Expected Return} = \text{Average}(R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \dots)$$

** Expected return:

Goal of reinforcement learning:

choose a policy $\pi(s) = a$ that will tell us what action a to take in state s so as to maximize the expected return.

Bellman Equations: $Q(s, a) = R(s) + \gamma \max_{\substack{\uparrow \\ 3}} Q(s', a')$

\downarrow \uparrow $2 \text{ or } 4$

* Discrete vs. Continuous State:

Data öncesi örneklerde tek bir dönen örneklerin temelî esas adıktır ve bunlar discrete örneklerdir. Fakat genelde havada helikopterin 3 boyuttaki durumu, hızı, yönü, yarpalaması vs. örneklidir veya bir tren statisini belirtmekten x, y düzleminde hız, mesafe (2.7) gibi bunlar continuous değerlerdir. Bu continuous değerler bir matris içinde tutulabilir.

Continuous state: truck $\rightarrow s = \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$

helicopter $\rightarrow s = \begin{bmatrix} x \\ y \\ z \\ \theta \\ \dot{\theta} \\ \text{roll pitch} \\ \text{yaw} \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\theta} \\ \dot{\dot{\theta}} \end{bmatrix}$

- x, y, z first state
- θ orientation
- $\dot{\theta}$ raw pitch
- w yaw
- $\dot{x}, \dot{y}, \dot{z}$ speed of directions
- $\dot{\theta}, \ddot{\theta}$ angular velocity

Lunar Lander:

Bir uçak aracının görevine imzaşının şartları nasıl kontrol edileceğini tartışın

actions: do nothing
 left thruster
 main thruster
 right thruster

} duruma göre alınlık/sıkılık
 atışyonlar standartı

$$s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

angle \rightarrow sola sağa rotasyon durumu
 left leg sitting \rightarrow sola sağa ayakta durum
 on the ground or not \rightarrow sağa ayakta durum

l, r / value 0 or 1 (binary)

Reward Functions:

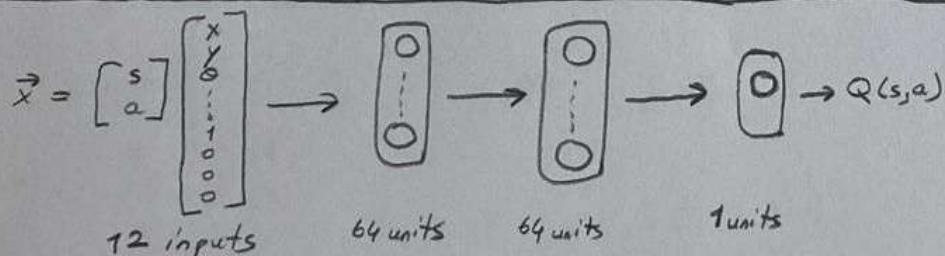
- Getting to landing pad: 100 - 140
- Additional reward for moving toward/away from pad
- Crash: -100
- Soft landing: +100
- Leg grounded: +10
- Fire main engines: -0.3
- Fire side thruster: -0.003

then learn a policy π that, given s

picks action $a = \pi(s)$ so as to maximize the return

and choose $\gamma = 0.985$ use a fairly large value for gamma

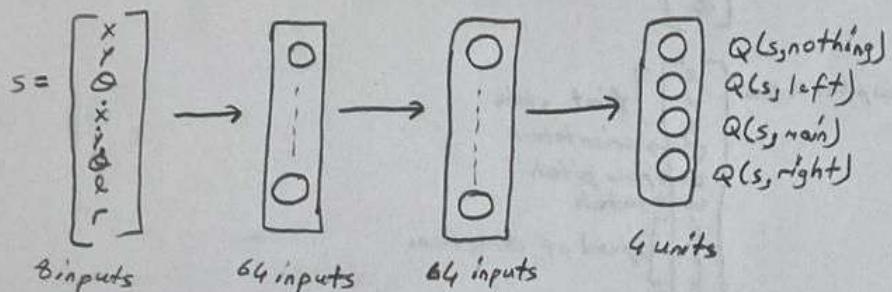
Deep Reinforcement Learning (different than supervised learning):



In a state s , use neural network to compute
 $Q(s, \text{nothing}), Q(s, \text{left}), Q(s, \text{main}), Q(s, \text{right})$

Pick the action a that maximizes $Q(s, a)$

Improved architecture:



ϵ -greedy policy:

In some states

option 1:

Pick the action a that maximizes $Q(s, a)$.

option 2:

with probability 0.95, pick the action a that maximizes $Q(s, a)$. Greedy, "exploitation"

with probability 0.05, pick an action a randomly. "Exploration"

ϵ -greedy policy ($\epsilon=0.05$)