Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow…

Category	Translation
Status	Done
Type	Book

Chapter 4. - Öyrənmə Modelləri

Bu vaxta qədər biz, Maşın Öyrənmə modelləri və onların öyrənmə alqoritmlərini daha çox qara qutu kimi təsəvvur edirdik. Əgər əvvəlki bölmələrdəki misalları edibsinizsə, sizə təəcüblü gələ bilərki mühərrikin altında nə baş verdiyini tam bilmədən nə qədər çox nəticə almışıq: reqresiya sistemlərini optimizasiya etmişik, rəqəmsal şəkil təsnifatı təkminləşdirmişik, hətta sıfırdan spam təsnifatçısını tərtib etmişik, bunların hamısı, onlar tam olaraq necə işlədiyini bilmədən etmişik. Həqiqətəndə, əksər hallarda bu implimentasiyaların detallarını bilməyə ehtiyac yoxdur.

Buna baxmayaraq, yaxşı anlayışınızın olması sizə uyğun modelin seçilməsi, düzgün öyrənmə alqoritminin istifadəsi, və düzgün hiper-parameterlərin (hyperparameters) seçilməsinə kömək edəcəkdir. Mühərrikin altında nə olduğunu başa düşmək əlavə olaraq, xətaları debug etməyə və onları daha səmərəli şəkildə analiz etməyə kömək edəcəkdir. Son olaraq isə, bu fəsildə danışacağımız mövzular neyron şəbəkələrinin (neural networks) başa düşülməsi, tərtib olunması və təlim olunması üçün vacib olacaqlar (Hissə II müzakirə olunacaq).

Bu fəsildə, ən sadə modellərdən biri olan, xətti reqressiya modeli (linear regression model) ilə başlacayıq. Biz modeli təlimləndirmək üçün iki ən çətin yolu müzakirə edəcəyik:

- “yaxın-forma” (closed-form) tənliyindən istifadə etməklə; hansı ki, təlim dəstinə ən uyğun gələn modelin, model parametrlərini birbaşa hesablayır (yəni, model parametrlər hansılar ki, təlim dəsti üzərində dəyər funksiyasını (cost function) azaldır).

- gradient eniş (GE) (gradient descent GD) adlanan iterativ optimizasiya yanaşmasından istifadə etməklə; hansı ki, tədricən təlim dəsti üzərində dəyər funksiyasını azalmaq üçün model parametrlərini düzəldir. Hansı ki, nəticədə birinci üsuldakı eyni parametrlər toplusuna yaxınlaşdırır. İrəlidə Hissə II-də, biz bir necə gradient eniş (GE) variantlarına baxaçıyıq, hansıları ki, neyron şəbəkələrini öyrənən zaman dəfələrlə istifadə edəcəyik. Misal üçün, toplu GE, mini-toplu GE, və stoxastik GE.

Sonra isə biz, polinom (polynomial) reqresiya, qeyri-xətti verilənlər toplusunu (nonlinear datasets) özündə əhatə edə biləcək mürəkkəb model ilə tanış olacıyıq. Polinom reqresiya modeli, xətti reqressiya modelindən daha çox parametrlərə malik olduğundan, o, təlim məlumatlarını həddən artıq uyğunlaşdırmağa daha çox meyllidir. Biz öyrənmə əyrilərindən (learning curves) istifadə edərək, bunun olub-olmadığını necə aşkar edəcəyimizi araşdıracağıq və sonra təlim dəstinə həddən artıq uyğunlaşma riskini azalda biləcək bir neçə nizamlama texnikasına baxacağıq.

Ən sonda isə, biz təsnifat tapşırıqları (classification tasks) üçün istifadə olunan daha iki modeli nəzərimizdən keçirəcəyik: logistik reqresiyası və softmax reqresiyası.

XƏDƏBRDARLIQ

Bu fəsildə kifayət qədər riyazi tənliklər istifadə olunacaq, hansılar ki, xətti cəbr və hesablamaların (calculus) əsas anlayışlarından istifadə edəcək. Bu tənlikləri başa düşmək üçün, vektor və matrislərin nə olduğunu; onları necə köçürmək, çoxaltmaq və tərsinə çevirmək; və qismən törəmə (partial derivatives) nədir bilməlisiniz. Əgər bu anlayışlarla tanış deyilsinizsə, zəhmət olmasa "Jupyter notebook" onlayn materiallar mövcuddur, hansılar ki, xətti cəbr və hesablamalar üzrə giriş dərslərini özündə əhatə edir. Əgər həqiqətən riyazi tənliklərə qarşı alergiyanız varsa, o zaman tənlikləri buraxa bilərsiniz, ümid edirik ki, sadəcə yazıları sizin başa düşməyiniz üçün kifayət edək.

Xətti Reqressiya

Fəsil 1-də biz, ən sadə reqressiyaya, həyat məmnunluğu reqressiyasına baxmışdıq:

$$life\_satisfaction = \theta_0 + \theta_1 \times GDP\_per\_capita$$

Bu model sadəcə *GDP_per_capita* daxiletmə xüsusiyyətinin xətti funksiyasıdır. θ_0 and θ_1 isə modelin parametrləridir.

Ümumi danışsaq, xətti model, daxiletmə xüsusiyyətlərininin (input features) çəkili cəmini hesablayaraq və qərəz termini adlanan (bias term) (bəzən kəsişmə termini (intercept term) də adlanır) sabiti ilə proqnoz verir, Tənlik 4-1 göstərildiyi kimi.

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

Tənlik 4-1. Xətti reqressiya modelinin proqnozu

Bu tənlikdə:

- $\hat{y}$ dəyəri proqnoz edir.
- n xüsusiyyətlərin sayıdır.
- x_i, i^{th}-nin xüsusiyyət dəyəridir.
- θ_j isə, qərəz termini θ_0 və çəkili xüsusiyyətlər olan $\theta_1, \theta_2, ..., \theta_n$ daxil olmaqla j^{th} modelinin parametridir.

Bunu isə, tənlik 4-2-də göstərildiyi kimi, vektorlaşdırılmış formadan (vectorized form) istifadə etməklə daha qısa şəkildə yazmaq olar.

$$\hat{y} = h_\theta(x) = \theta \cdot x$$

Tənlik 4-2. Xətti reqressiya modelinin proqnozu (vektorlaşdırılmış forma)

Bu tənlikdə:

- h_θ - θ model parametrlərini istifadə edən hipotezis (hypothesis) funksiyasıdır.
- θ - modelin vektor parametridir, hansı ki, özündə qərəz termini θ_0 (bias term) və çəkili xüsusiyyətlər olan $\theta_1, \theta_2, ..., \theta_n$ əhatə edir.
- x - nümunənin xüsusiyyət (feature) vektorudur, hansı ki, özündə x_0 to x_n, harda ki, x_0 bərabərdir birə (1).
- $\theta \cdot x$ ifadəsi, θ və x vektorların nöqtəli hasilidir (dot product), hansı ki, $\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$.

QEYD

Maşın öyrənmədə (machine learning), vektorlar adətən "sütun" vektorları (column vectors) kimi təmsil olunur, hansı ki, tək sütun 2D massivlərdir (arrays). Əgər, θ və x sütun vektorlardırsa, onda proqnoz $\hat{y} = \theta^\top x$, harda ki, $\theta^\top$, θ-nın köçürülməsidir (transpose) (sütun vektor əvəzinə sətir vektor) və $\theta^\top x$ isə, $\theta^\top$ və x matriks vurulmasıdır. Əlbəttə bu eyni proqnozdur, amma istisna olaraq, o, indi skalyar (scalar) əyərdən daha çox tək-hüceyrə matriks kimi təqdim olunur. Bu kitabda, nöqtəli hasililəri və matriks vurulması arasında keçidi yığışdırmaq üçün bu annotasiyadan istifadə edəcəm.

Aydındı, bu xətti reqresiya modelidir, bəs biz bunu necə təlim etməliyik? Gəlin yadımıza salaq, modeli təlim etmək, onun parameterlərini elə sazlamaq deməkdir ki, model təlim dəstinə ən uyğunu olsun. Bunun üçün, biz birinci olaraq model təlim datasına nə dərəcədə yaxşı (və ya pis) uyğun gəldiyini ölçməkdir. FƏSİL 2-də, biz gördük ki, reqressiya modelinin ən ümumi performans göstəricisi kök orta kvadrat xətadır (Tənlik 2-1). Buna görə, xətti reqresiya modelini təlim etmək üçün, biz θ dəyərini tapmalıyıq, hansı ki, RMSE (root mean square error) minimallaşdıracaq. Praktikada, orta kvadrat xətasını (mean squared error MSE) minimallaşdırmaq daha asandır, nəyin ki, kök orta kvadrat xətasını (root mean squared error RMSE), bu isə eyni nəticəyə getirib çıxarır (çünki müsbət fuksiyanı minimallaşdıran dəyər, kvadrat kökü də minimallaşdırır).

XƏDƏBRDARLIQ

Öyrənmə alqoritmləri, təlim zamanı çox vaxt müxtəlif itki funksiyalarını (loss function) optimallaşdırır, nəyinki, son modeli qiymətləndirmək üçün istifadə olunan performans ölçülərini. Ümumiyyətlə, ona görə ki, funksiyaları optimizasiya etmək daha asandır, və/vəya ona görə ki, təlim zamanı lazım olan əlavə şərtlərə malik olması ilə əlaqədardır (məsələn, nizamlanma üçün). Yaxşı performans göstəricisi, yekun biznes obyektivinə ən yaxın olandır. Yaxşı təlim itkisini isə optimizasiya etmək daha asandır və göstərici ilə güclü şəkildə əlaqələndirilir. Misal üçün, klassifikatorlar, tez-tez log itkisi olan xərc funksiyasından istifadə etməklə təlimləndirilir (daha sonra görəcəyik), lakin, "dəqqlik/qeri çağırış" vasitəsiylə qiymətləndirilir. Log itkisini (log loss) minimallaşdırmaq asandır, və bunu etməklə, "dəqqlik/qeri çağırış" (precision/recall) adətən yaxşılaşacaqdır.

Öyrənmə çoxluğu $\mathbf{X}$ üzərində xətti reqresiya fərziyyəsi h_θ-nin orta kvadrat xətasını Tənlik 4-3 vasitəsilə hesablanır.

$$\mathrm{MSE}\left(\mathbf{X}, h_\theta\right) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta^\top \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

Tənlik 4-3. Xətti reqressiya modeli üçün orta kvadrat xətası xərc fuksiyası

Əksər notasiyalar (notation) Fəsil 2 mövcüdur ("Notasiyalar" bax). Buradakı tək fərq isə, biz h əvəzinə h_θ yazmışıq, hansı ki, daha açıq şəkildə izah edir ki, model, vektor θ vasitəsiylə parametrləşdirilmişdir. Notasiyanı sadələşdirməkdən ötrü, biz $\mathrm{MSE}\left(\mathbf{X}, h_\theta\right)$ əvəzinə $\mathrm{MSE}\left(\theta\right)$ kimi yazacayıq.

Normal Tənlik

θ-nin dəyərini tapmaq üçün, hansı ki, orta kvadrat xətasını minimallaşdırır, "yaxın-forma" (closed-form) həlli mövcuddur - başqa cür desək, bir başa nəticəni verən riyazi tənlik. Bu Normal Tənlik (Normal Equation) adlandırılır (Tənlik 4-4).

$$\hat{\theta} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

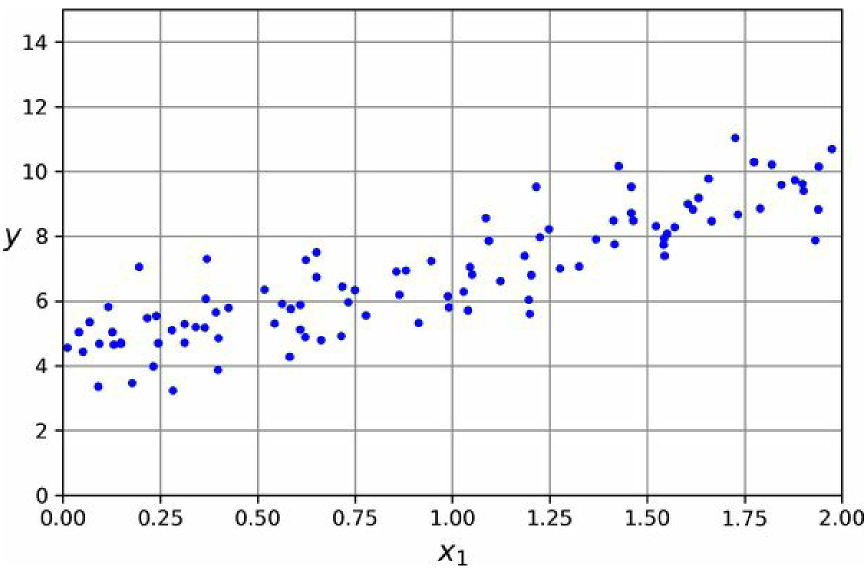
Tənlik 4-4. Normal tənlik

Bu tənlikdə:

- $\hat{\theta}$ - θ-nın dəyəri, hansı ki, xərc funksiyasını minimallaşdırır.
- y, $y^{(1)}$-dən $y^{(m)}$-ə kimi özündə saxlayan hədəf dəyərlərin vektorudur.

Gəlin tənliyi test etmək üçün xətti görünüşlü (linear looking) data generate edək (Şəkil 4-1).

```
import numpy as np

np.random.seed(42) # to make this code example reproducible
m = 100 # number of instances
X = 2 * np.random.rand(m, 1) # column vector
y = 4 + 3 * X + np.random.randn(m, 1) # column vector
```



Şəkil 4-1. İxtiyari şəkildə generate olunmuş xətti data çoxluq.

İndi isə, gəlin normal tənlikdən istifadə edərək $\hat{\theta}$ hesablayaq. Matrisin tərsini hesablamaq üçün "NumPy" kitabxanasının xətti alqebra modulundan $inv()$ fuksiyasından və matrisin hasilini hesablamaq üçün $dot()$

funksiyasından istifadə edəcəyik.

```
from sklearn.preprocessing import add_dummy_feature

X_b = add_dummy_feature(X) # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
```

QEYD

@ operatoru matrisin hasilini hesablamaq üçün istifadə olunur. Əgər A və B, NumPy massivləridirsə, o zaman A @ B bərabərdir np.matmul(A, B) ifadəsinə. Çoxlu digər kitabxanalar, TensorFlow, PyTorch və JAX @ operatorunu dəstəkliyir. Ancaq, xalis (pure) Python massivlər (i.e. siyahıların siyahıları) @ operatorundan istifadə edə bilmir.

$y = 4 + 3_{x_1} + Gaussian\ noise$ funksiyasını biz datanı yaratmaqdan ötrü istifadə etmişik. Nəticəyə baxaq:

```
>>> theta_best
array([[4.21509616],
       [2.77011339]])
```

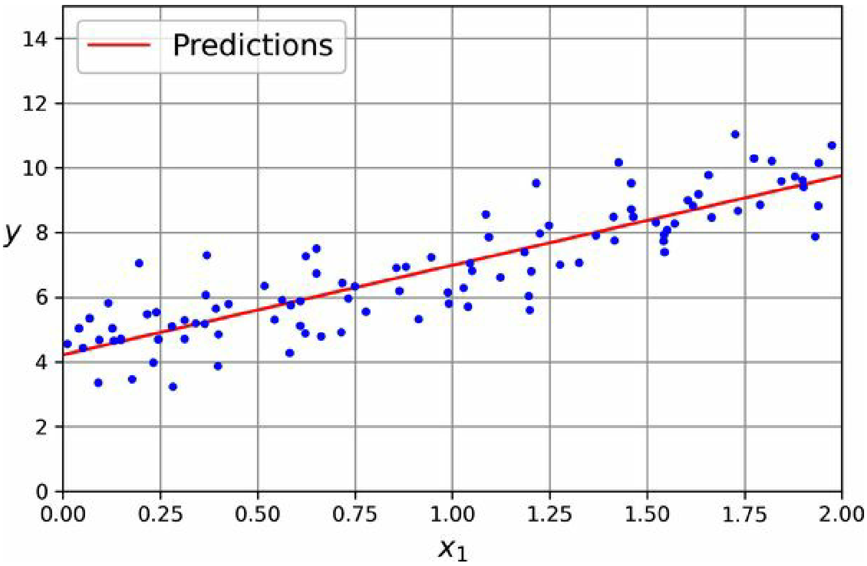
Biz $\theta = 4.215$ və $\theta_1 = 2.770$ əvəzinə $\theta = 4$ və $\theta_1 = 2$ olmasını ümid edərdik. Kifayət qədər yaxındı, lakin səs-küy (noise) orijinal funksiyanın dəqiq parametrlərini bərpasını qeyri-mümkün etdi. Məlumat çoxluğu nə qədər kiçik və ses-küylü olsa, bir o qədər çətin əmələ gəlir.

İndi isə biz, $\hat{\theta}$ istifadə etməklə proqnozlaşdıra bilərik:

```
>>> X_new = np.array([[0], [2]])
>>> X_new_b = add_dummy_feature(X_new)  # add x0 = 1 to each instance
>>> y_predict = X_new_b @ theta_best
>>> y_predict
array([[4.21509616],
       [9.75532293]])
```

Gəlin bu modelin proqnozlarını tərtib edək(Şəkil 4-2):

```
import matplotlib.pyplot as plt

plt.plot(X_new, y_predict, "r-", label="Predictions")
plt.plot(X, y, "b.")
[...]  # beautify the figure: add labels, axis, grid, and legend
plt.show()
```



Şəkil 4-2: Xətti reqressiya modelinin proqnozları

Scikit-Learn istifadə edərək xətti reqressiyanın həyata keçirilməsi nisbətən sadədir:

```
>>> from sklearn.linear_model import LinearRegression
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([4.21509616]), array([[2.77011339]]))
>>> lin_reg.predict(X_new)
array([[4.21509616],
       [9.75532293]])
```

Diqqət yetirin ki, Scikit-Learn qərəzli termini (intercept_) xüsusiyyət çəkilərindən (coef_) ayırır. “LinearRegression” sinifi isə, scipy.linalg.lstsq() funksiyasına əsaslanır (ad “ən kiçik kvadratlar” deməkdir) və bir başa çağırıla bilər:

```
>>> theta_best_svd, residuals, rank, s = np.linalg.lstsq(X_b, y, rcond=1e-6)
>>> theta_best_svd
array([[4.21509616],
       [2.77011339]])
```

Bu funksiya $\hat{\theta} = \mathbf{X}^{+}\mathbf{y}$, hardakı, $\mathbf{X}$-ın psevdotərsi (pseudoinverse) $\mathbf{X}^{+}$-dir (dəqiq olsaq, Moor-Penrose tərsi). Psevdotərsi hesablamaq üçün, siz bir başa np.linalg.pinv() istifadə edə bilərsiniz:

```
>>> np.linalg.pinv(X_b) @ y
array([[4.21509616],
```

```
        [2.77011339]])
```

Psevdotərsin özü $\mathbf{X}$ təlim çoxluğu matrisini üç $\mathbf{U}\ \mathbf{\Sigma}\ \mathbf{V}_{\top}$ matrisin matris hasilinə parçalaya bilən tək dəyər parçalanması (singular value decomposition) adlanan standart matris faktorizasiyası texnikasından istifadə etməklə hesablanır (bax, numpy.linalg.svd()). Psevdotərs $\mathbf{X}^{+} = \mathbf{V}\mathbf{\Sigma}^{+}\mathbf{U}^{\top}$ ilə hesablanır. $\mathbf{\Sigma}^{+}$ matrisini hesablamaq üçün, alqoritm $\mathbf{\Sigma}$ götürür və kiçik hədd dəyərindən kiçik bütün dəyərlərə sıfır mənimsədir, sonra bütün sıfırdan fərqli olan dəyərli onların tərsi ilə əvəz edir və yekunda yekun matrisi köçürür. Bu yanaşma normal tənliyi hesablamaqdan daha səmərəlidir və əlavə olaraq, o, xüsusi halları gözəl şəkildə idarə edir: həqiqətən də, normal tənlik $\mathbf{X}^{\top}\mathbf{X}$ matrisi tərs çəvrilə bilən deyilsə işləməyə bilər (yəni, tək), məsələn, əgər $m < n$ və ya bəzi xüsusiyyətlər lazımsızdır amma psevdotərsi həmişə müəyyəndir.

Hesablama mürəkkəbliyi

$\mathbf{X}^{\top}\mathbf{X}$ matrisin tərsini normal tənlik hesablayır, hansı ki, $(n+1)\times(n+1)$ matrisidir (n - xüsusiyyətlərin sayıdır). Tətbiqdən asılı olaraq, matris tərslinə çevirməsinin hesablama mürəkkəbliyi adətən $O(n^{2.4})$-dən $O(n^{3})$ kimidir. Başqa sözlə, xüsusiyyətlərin sayını 2 dəfə artırsanız, hesablama vaxtını təxminən $2^{2.4} = 5.3$ ilə $2^{3} = 8$ vurursunuz.

Scikit-Learn-in “LinearRegression” sinifi tərəfindən istifadə olunan tək dəyər parçalanması $O(n^{2})$ qədərdir. Xüsusiyyətlərin sayısını ikiqat artırsanız, hesablama vaxtı təxminən 4 dəfə artırarsız.

XƏBƏRDARLIQ

Əgər xüsusiyyətlərin sayı çox böyüyürsə (məsələn, 100,000), o zaman həm normal tənlik həmdə ki tək dəyər parçalanması çox yavaşlayır. Müsbət tərəfi isə, hər ikisi, təlim çoxluğundakı nümunələrin sayısını nəzərə almaqla xəttidir (onlar, $O(m)$), ona görə də, yaddaşa sığmaq şərti ilə geniş təlim çoxluğunu səmərəli idarə edirlər.

Həmçinin, xətti reqressiya modelinizi təlimləndirdikdən sonra (normal tənlik və ya hər hansı digər alqoritm vasitəsilə), proqnozlama daha sürətli baş verir: hesablama mürəkkəbliyi həm proqnoz vermək istediyiniz nümunələrin sayına həm də xüsusiyyətlərin sayına nəzərən xəttidir. Başqa sözlə, ikiqat çox nümunə ilə (və ya ikiqat çox xüsusiyyət ilə) proqnozlaşdırmaq ikiqat daha çox vaxt aparacaq.

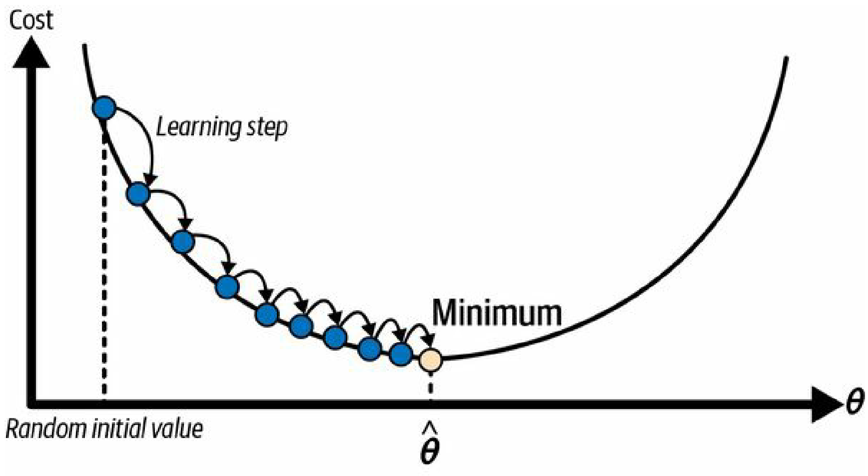
İndi isə, biz, xətti reqressiya modelini təlimləndirməyin ferqli üsulunu nəzərdən keçirəcəyik ki, bu, çoxlu xüsusiyyətlərin və ya yaddaşa sığdırmaq üçün çoxlu təlim nümunələrinin olduğu hallar üçün daha uyğundur.

Qradient Eniş

Qradient Eniş, geniş problemlər çeşidinə optimal həllin tapmağa qadir olan ümumi optimallaşdırma alqoritmidir. Qradient enişin ümumi ideası, dəyər funksiyasını minimallaşdırmaq üçün təkrar olaraq parametrləri çızmaqdır.

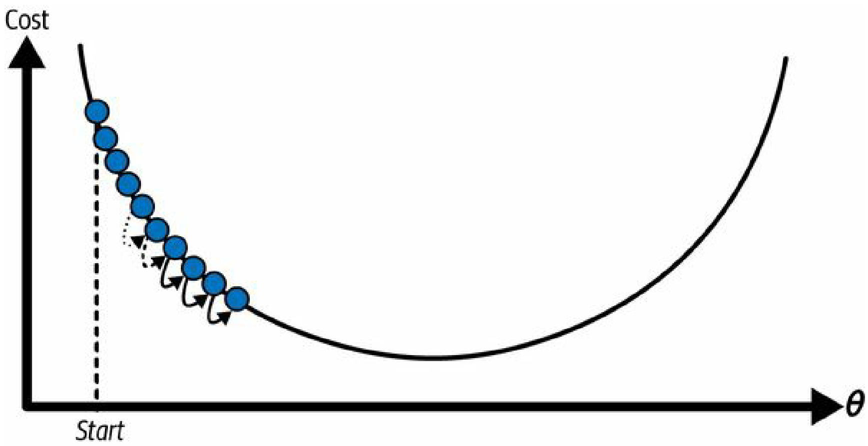
Təsəvvür edin ki, siz dağda sıx dumanda itmisiniz, və siz yanlız ayağınızın altındakı yamacı hiss edirsiniz. Vadinin aşağısına tez çatmaq üçün ən yaxşı strategiya, ən dik yamac istiqamətində aşağı ənməkdir. Qradient eniş məhz bunu edir: o, θ vektor parametr ilə bağlı olan xəta funksiyasının daxili (local) qradientini ölçür, və azalan qradient istiqamətində hərəkət edir. Qradient sıfıra çatdırda isə, minimuma çatmış olursunuz!

Praktikada, siz θ vektor parametrini ixtiyari dəyərlərlə doldurmaqla başlıyırsınız (buna, ixtiyari inisializasiya (random initialization) deyirlər). Sonra, alqoritm minimuma yaxınlaşana qədər, hər bir addımda dəyər funksiyasını (məsələn, MSE) azaltmağa çalışaraq, tədricən təkmilləşdirirsiniz (Şəkil 4-3).



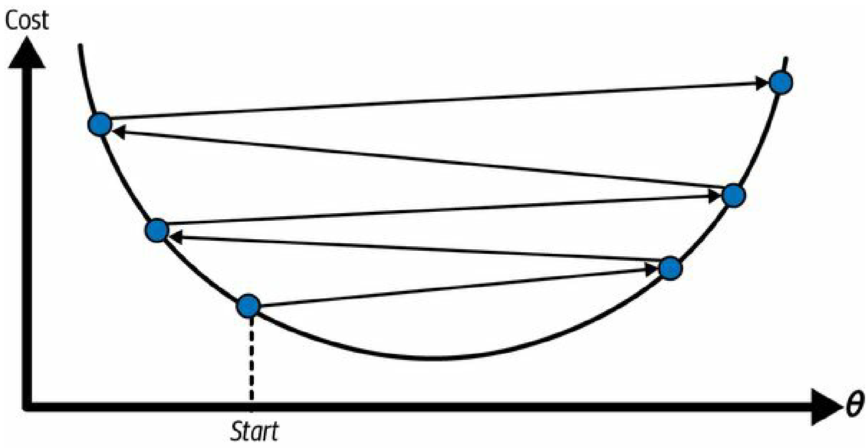
Şəkil 4-3: Bu qradient enişdə, modelin parametrləri ixtiyari olaraq inisializasiya olub və dəyər funksiyasını minimuma endirmək üçün dəfələrlə dəyişir; öyrənmə addımının ölçüsü dəyər funksiyasının enişi ilə düzmütənasibdir; ona görədə dəyər minimuma yaxınlaşdıqca addımlar tədricən kiçilir.

Qradient enişin ən önəmli parametri, öyrənmə dərəcəsi ilə təyin olunan addımların sayıdır. Əgər öyrənmə dərəcəsi çox kiçikdirsə, onda alqoritm yaxınlaşmaq üçün çoxlu iterasiyalardan keçməli olacaq ki, bu da çox vaxt aparacaq (Şəkil 4-4).



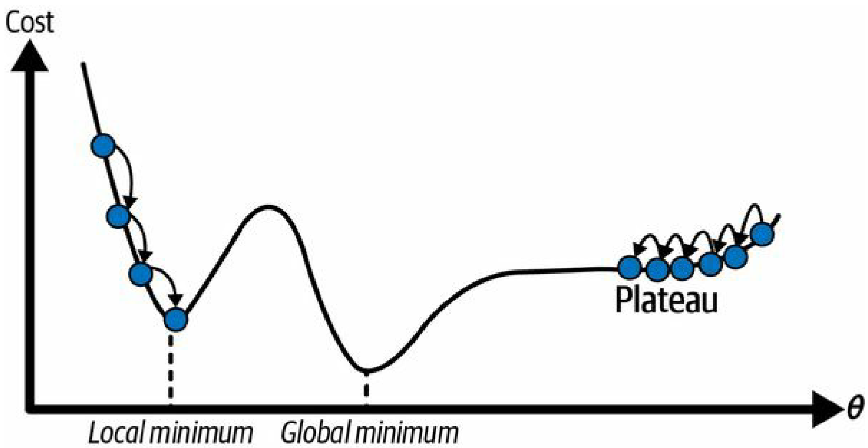
Şəkil 4-4. Öyrənmə dərəcəsi çox kiçikdir.

Digər tərəfdən, əgər öyrənmə dərəcəsi çox yuxarıdırsa, o zaman siz çökəklikdən keçib o bir tərəfə, bəlkə də əvvəlkindən daha yüksəklərə qalxa biləcəksiniz. Bu isə alqoritmi daha da yüksək dəyərlərlə normadan kənara çıxmağa səbəb olub, yaxşı həllin tapılmamasına səbəb ola bilər (bax, Şəkil 4-5).



Şəkil 4-5. Öyrənmə dərəcəsi çox yüksəkdir.

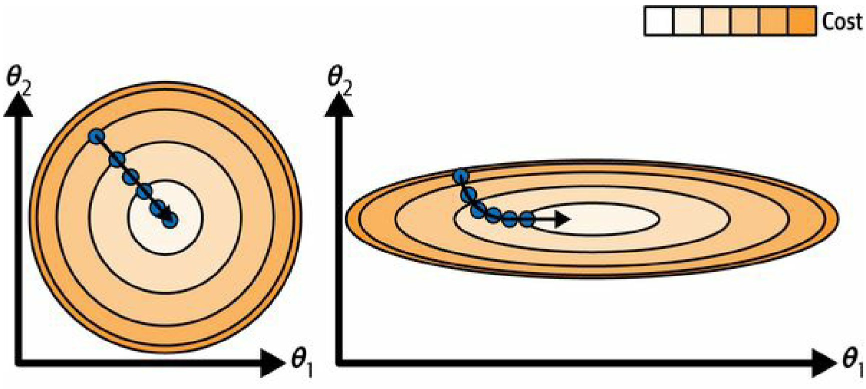
Bundan əlavə, hər bir dəyər fuksiyasının görünüşü ideal kasa görünüşü kimi gözəl olmaya bilər. Çuxurlar, silsilələr, yaylalar və hər cür qeyri-müntəzəm ərazilər ola bilər, bu da minimuma yaxınlaşmanı çətinləşdirir. Şəkil 4-6, qradient eniş ilə bağlı əsas iki çətinliyi göstərir. Əgər ixtiyari inisializasiya alqoritmi soldan başlasa, o zaman, daxili minimumda cəmlənəcək, hansı ki, qlobal minimum qədər yaxşı deyil. Əgər o sağdan başlasa, onda yaylağı keçmək üçün çox uzun vaxt sərf edəcək. Və əgər o çox erkən dəyansa, o zaman heç vaxt global minimuma çata bilməyəcək.



Şəkil 4-6. Qradient enişin tələləri

Xoşbəxtlikdən, xətti reqressiya modeli üçün “orta kvadrat xətası” funksiyası qabarıq (convex) funksiyaya çevrilir, yəni əyridə hər hansı iki nöqtəni seçsəniz, onları birləşdirən xətt seqmenti heç vaxt əyridən aşağı olmayacaqdır. Bu o deməkdir ki, burada local minimum yoxdur, sadəcə qlobal minimum mövcuddur. Bu həmdə, heç vaxt kəskin dəyişməyən enişli funksiyadır. Bu qeyd olunan iki faktın böyük nəticəsi var: qradient eniş, özünün qlobal minimuma nə qədər yaxın istəsə, o qədər yaxınlaşmasını təmin edir (əgər kifayət qədər gözləsəz və öyrənmə dərəcəsi çox yuxarı deyilsə).

Dəyər funksiyası kasavari formaya malik olsa da, əgər xüsusiyyətlər fərqli miqyasları varsa, o, uzunsov kasavari formada ola bilər. Şəkil 4-7, 1 və 2-ci xüsusiyyətləri eyni miqyasa sahib olan (sol tərəf) və 1ci xüsusiyyətin 2-cidən daha kiçik dəyərlərə sahib olan (sağ tərəf) təlim dəstinin qradient enişini göstərir.



Gördüyümüz ki, sol tərəfdə qradient eniş alqoritmi birbaşa minimuma doğru gedir, bununla da ona tez çatır, halbuki, sağ tərəfdə o birinci, qlobal minimum istiqamətinə, demək olar ki ortoqonal vari istiqamətdə hərəkət edir və sonda, demək olar ki vadidən aşağı uzun bir yürüşlə yekunlaşır.

XƏBƏRDARLIQ

Qradient enişin istifadəsi zamanı, bütün xüsusiyyətlərin eyni miqyasa malik olduğunu təmin olunmalıdır (məsələn, Scikit-Learn-nun StandardScalesr sinifi), əks halda bir nöqtədə cəmləşmək üçün daha çox vaxt sərf olunacaq.

—

Bu diaqram həmçinin göstərir ki, modeli telimləndirmək, dəyər funksiyasını minimallaşdıran (təlim dəsti üzərində) model parametrlərin birləşməsini axtarmaq deməkdir. Bu, modelin parametr vəzasında axtaşırdır. Modelin nə qədər çox parametri varsa, vəzanın bir o qədər çox ölçüsü olacaq, bu da axtarışı çətinləşdirir: 300 ölçülü ot tayasından iynə axtarmaq, 3 ölçülüdən qat qat daha çətindir. Xoşbəxtlikdən, xətti reqresiyası halında dəyər fuksiyası qabarıq olduğundan, iynə kasanın dibindədir.

Toplu Qradient Enişi

Qradient enişini tətbiq etmək üçün, hər bir θ_j model parametri ilə bağlı olan dəyər fuksiyasının qradientini hesablamalısınız. Başqa sözlə desək, θ_j -i bir az dəyişdirsəniz, dəyər fuksiyasının nə qədər dəyişəcəyini hesablamalısınız. Buna, *qismən törəmə* deyilir. Bu, "Şərqə baxaraq, ayağımın altındakı dağın yamacı nədir" sualına bənzəyir? və sonra eyni sualı şimala baxaraq vermək (və s., əgər üçdən çox ölçülü bir kainatı təsəvvür edə bilsəniz, bütün digər ölçülər üçün də bu sualı verə bilərsiniz). Tənlik 4-5, $\frac{\partial}{\partial \theta_j} \mathrm{MSE}(\theta)$ kimi qeyd olunub, θ_j nəzərən orta kvadrat xətasının qismən törəməsini hesablayır.

$$\frac{\partial}{\partial \theta_j} \mathrm{MSE}(\theta) = \frac{2}{m} \sum_{i=1}^{m} \left(\theta^\top \mathbf{x}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

Tənlik 4-5. Dəyər funksiyasının qismən törəməsi

Ayrı-ayrılıqda qismən törəməni hesablamaq əvəzinə, siz, Tənlik 4-6-dan istifadə etməklə hamısını bir anda hesablaya bilərsiniz. $\nabla_\theta \, \mathrm{MSE}(\theta)$ kimi qeyd olunan qradient vektoru, dəyər funksiyasının bütün qismən törəmələrini özündə cəmləşdirir (hər bir model parametr üçün).

$$\nabla_\theta \operatorname{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial\theta_0} \operatorname{MSE}(\theta) \\ \frac{\partial}{\partial\theta_1} \operatorname{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial\theta_n} \operatorname{MSE}(\theta) \end{pmatrix} = \frac{2}{m}\mathbf{X}^\top(\mathbf{X}\theta - \mathbf{y})$$

Tənlik 4-6. Dəyər funksiyasının qradient vektoru

XƏBƏRDARLIQ

Diqqət yetirsək görərik ki, bu tənlik, hər qradient eniş addımında $\mathbf{X}$ tam təlim dəsti üzərində hesablama aparır! Buna görə də, bu tənlik "toplu qradient eniş" adlandırılır: o, hər bir addımda təlim dəstinin toplusu istifadə edir (əslində, "tam qradient eniş" daha doğru adlandırma olardı). Nəticədə, o çox böyük təlim dəstləri üçün olduqça çox vaxt sərf edir (biraz sonra, nir neçə daha sürətli qradient eniş alqoritmlərinə baxacağıq). Bununla belə, qradient eniş, xüsusiyyətlərin sayı ilə daha yaxşı ölçülür; Əgər yüz minlərlə xüsusiyyətlər varsa, normal tənlik vəya SVD parçalanmasını istifadə etməkdənsə qradient enişi istifadə etsək, o zaman xətti reqresiya modelini daha tez təlim etmiş olarıq.

—-

Yoxuşu göstərən qradient vektorunuz olduqda, aşağı ənmək üçün əks istqamətdə getmək kifayətdir. Bu, θ-dan $\nabla_\theta \operatorname{MSE}(\theta)$-nı çıxmaq deməkdir. Burada öyrənmə sürəti η işə düşür: eniş addımların ölçüsünü müəyyən etmək üçün qradient vektorunu η ilə vurun (Tənlik 4-7).

$$\theta^{(\text{next step })} = \theta - \eta\nabla_\theta \operatorname{MSE}(\theta)$$

Tənlik 4-7. Qradient eniş addımı

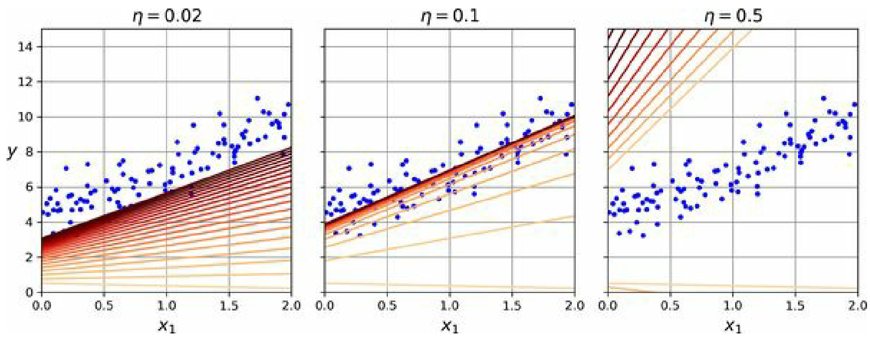
Gəlin bu alqoritmin tətbiğinə baxaq:

```
eta = 0.1  # learning rate
n_epochs = 1000
m = len(X_b)  # number of instances

np.random.seed(42)
theta = np.random.randn(2, 1)  # randomly initialized model parameters

for epoch in range(n_epochs):
    gradients = 2 / m * X_b.T @ (X_b @ theta - y)
    theta = theta - eta * gradients
```

Bu çox da çətin deyildi! Təlim dəsti üzərində hər bir iterasiya epoch (dövr) adlanır. Gəlin, "theta"-nın nəticəsinə baxaq:

```
>>> theta
array([[4.21509616],
       [2.77011339]])
```

Normal tənliyin tapdığı ilə eyni nəticədir! Qradient eniş mükəmməl işlədi. Bəs, əgər biz fərqli öyrənmə sürətini (eta) istifadə etsək ne olardı? Şəkil 4-8, üç fərqli öyrənmə sürətindən istifadə etməklə, qradient enişin ilk 20 addımı təsvir edir. Hər bir sahənin altındakı xətt təsadüfi başlanğıc nöqtəsini təsvir edir, sonra hər bir epoach (dövr) daha da qaranlıq xətt ilə təsvir olunur.



Solda, öyrənmə dərəcəsi çox aşağıdır: alqoritm ən axırda nəticəyə gəlib çatacaq, lakin bu, çox zaman tələb edəcək. Ortada, öyrənmə dərəcəsi kifayət qədər yaxşı görünür: cəmi bir neçə epoach (dövr) ərzində o, artıq həllə yaxınlaşmışdır. Sağda isə, öyrənmə dərəcəsi çox yüksəkdir: alqoritm fərqlənir, hər yerə tullanır və əslində hər addımda həllədn daha da uzaqlaşır.

Yaxşı öyrənmə sürətini tapmaq üçün, siz grid (şəbəkə) axtarışından istifadə edə bilərsiniz (Fəsil 2). Bununla belə, grid axtarışı uzun çəkən modelləri aradan qaldırsın deyə epoach(dövr)-ların sayını məhdudlaşdıra bilərsiniz.

Sizə maraqlı gələ bilər epoach-ların (dövr) sayını necə təyin etmək olar. Əgər çox aşağıdırsa, alqoritm dəyandıqda belə, siz optimal həlldən çox uzaq olacaqsınız; lakin əgər çox yüksək olsa, o zaman, vaxt itirəcəksiniz, o vaxt ki, modelin parametrləri artıq dəyişmir. Sadə həll yolu, çoxlu sayda epoach-ları (dövr) təyin etməkdir, lakin qradient vektoru kiçik olduqda alqoritmi dayandırmaqdır, - yəni, onun norması kiçik ϵ ədədindən (tolerantlıq adlanır) daha kiçik olduqda - çünki bu, qradient enişin minimuma çatdıqda baş verir.

CONVERGENCE DƏRƏCƏSİ

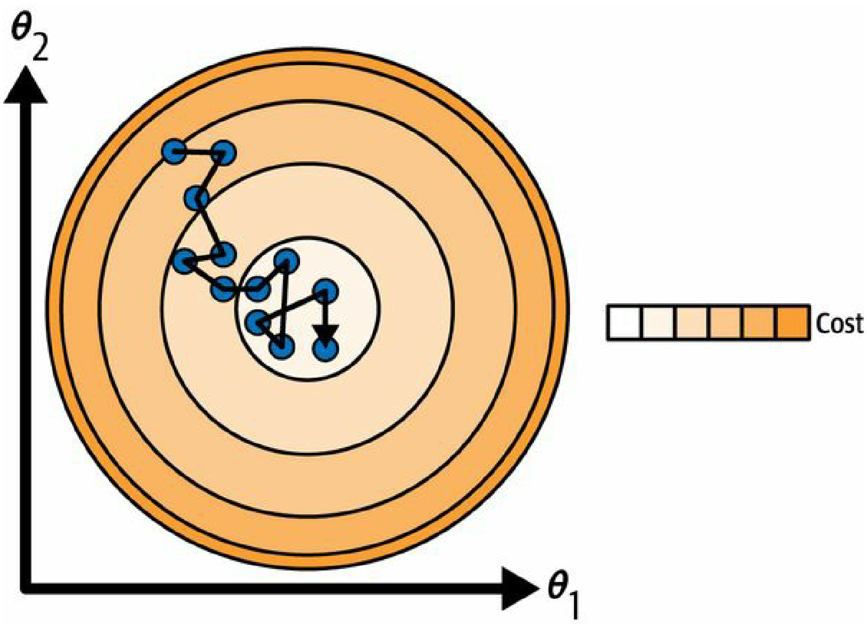
Dəyər funksiyası qabarıq olduqda və onun mailliyi kəskin dəyişmədikdə (orta kvadrat xətasının dəyər funksiyası üçün olan halda), sabit öyrənmə dərəcəsi ilə toplu qradient enişi optimal həllə yaxınlaşacaq, lakin bir müddət gözləməli olacaqsınız: dəyər funksiyasının formasından asılı olaraq, ϵ diapazonunda optimala çatmaq üçün $O(\frac{1}{\epsilon})$ qədər iterasiya lazım ola bilər. Daha dəqiq bir nəticə eldə etmək üçün tolerantlığı 10 dəfə bölsəniz, onda bu, alqorithmin təxminən 10 dəfə daha çox icra olunmasına gəlib çıxara bilər.

—

Stoxastik Qradient Eniş

Toplu Qradien Enişin problemi odur ki, onun hər addımda qradientləri hesablamaq üçün bütün təlim dəstindən istifadə etməsidir ki, bu da təlim çoxluğu böyük olduqda onun çox yavaşlamasına gəlib çıxarır. Əks tərəfdə is, stoxastik qradient eniş hər addımda təlimdə ixtiyari bir nümunə götürür, və yalnız həmin nümunə əsasında qradientləri hesablayır. Aydındır ki, tək tək bir nümunə üzərində əməliyyat yerinə yetirmək alqoritmi daha sürətli edir, çünki o, hər addımda çox az məlumat idarə edir. Həmçinin, o, böyük təlim dəstləri üzərində təlim keçməyə imkak verir, çünki hər addımda yaddaşda yanlız bir nümunə saxlamaq kifayət edir (Stoxastik Qradient Eniş "out-of-core" alqoritmi kimi də tətbiq oluna bilər; Fəsil 1-ə bax).

Digər tərəfdən, onun stoxastik (yəni, ixtiyari) təbiətinə görə, bu alqorithm, toplu qradient enişdən daha az müntəzəmdir: minimuma çatana qədər yavaş-yavaş azaltmaq əvəzinə, dəyər funksiyası yalnız orta hesabla azalaraq, yuxarı və aşağı sıçrayacaqdır. Vaxt keçdikcə o, minimuma çox yaxınlaşacaq, lakin ora çatdıqdan sonra dayanmayacaq, sıçramağa davam edəcək (Şəkil 4-9). Bir vaxt, alqoritm dayandıqdan sonra, yekun parametrin dəyəri yaxşı olacaq, lakin optimal olmayacaqdır.



Şəkil 4-9. Stokastik gradient eniş ilə, hər bir təlim addımı, toplu qradient enişin istifadəsindən daha sürətli, amma həm də daha çox stoxastik (ixtiyari) olur.

Dəyər funksiyası çox qeyri-müntəzəm olduqda (Şəkil 4-6 olduğu kimi), bu, əslində alqoritmin daxili minimumdan çıxmasına kömək edə bilər, və beləliklə, stoxastik qradient eniş qlobal minimumu tapmaq şansı toplu qradient enişdən daha yüksəkdir.

Buna görə də, daxili optimadan qaçmaq üçün ixtiyarilik yaxşıdır, amma eyni anda pisdir, çünki bu, alqoritmin heç vaxt minimumda yerləşəməyəcəyinin mənasına gəlir. Bu dilemmanın həlli yolu odur ki, öyrənmə dərəcəsini tədricən azaltmaqdır. Addımlar böyükdən başlayır (bu, sürətli proqress eldə etməyə və daxili minimumdan xilas olmağa kömək edir), sonra isə, addımlar getdikcə kiçilir və alqoritmin qlobal minimumda qərarlaşmasına imkan verir. Bu proses, ərimiş metalın yavaş-yavaş soyudulduğu "yumşalma metallurgiyası" (metallurgy of annealing) prosesindən ilhamlanan "simulyasiya edilmiş yumşalma" alqoritminə bənzəyir. Hər iterasiyada öyrənmə dərəcəsini təyin edən funksiyaya "öyrənmə cədvəli" deyilir. Əgər öyrənmə dərəcəsi çox tez azalarsa, siz daxili minimumda ilişib qala bilərsiniz və ya hətta minimuma qədəndə, yolun yarısında donub qala bilərsiniz. Əgər öyrənmə dərəcəsi çox yavaş azalarsa, o zaman, siz uzun müddət minimumun ətrafında fırlana bilərsiniz, və əgər təlimi çox tez dayandırsanız, bu, optimal olmayan həll yolu ilə nəticələ bilər.

Bu kod, sadə öyrənmə cədvəlindən istifadə edərək stoxastik gradient enişini heyata keçirir:

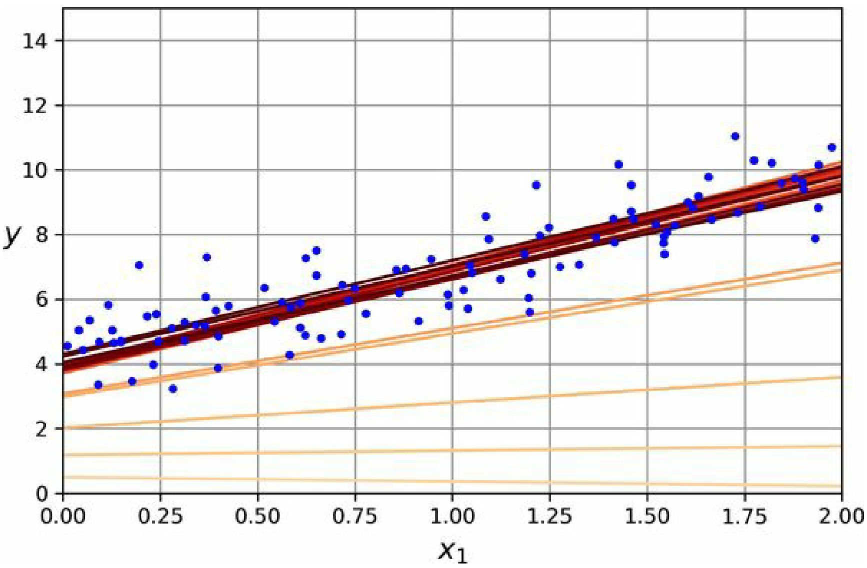
```
n_epochs = 50
t0, t1 = 5, 50  # learning schedule hyperparameters

def learning_schedule(t):
    return t0 / (t + t1)

np.random.seed(42)
theta = np.random.randn(2, 1)  # random initialization

for epoch in range(n_epochs):
    for iteration in range(m):
        random_index = np.random.randint(m)
        xi = X_b[random_index : random_index + 1]
        yi = y[random_index : random_index + 1]
        gradients = 2 * xi.T @ (xi @ theta - yi)  # for SGD, do not divide by m
        eta = learning_schedule(epoch * m + iteration)
        theta = theta - eta * gradients
```

Şərti olaraq, biz her bir dövrdə "m" iterasiyası qədər dövr edirik; hər bir dövr, "epoach" adlanır. Toplu qradient enişin kodu öyrənmə dəsti üzərində 1000 dəfə dövr etsə də, bu kod öyrənmə dəsti üzərində sadəcə 50 dövr edir və olduqca yaxşı nəticəyə gəlib çıxır:

```
>>> theta
array([[4.21076011],
       [2.74856079]])
```

Şəkil 4-10, təlimin ilk 20 addımını göstərir (addımların nə qədər nizamsız olduğuna diqqət yetirin).

Nəzərə alın ki, dəyərlər ixtiyari seçildiyi üçün, dövr ərzində bəzi dəyərlər bir neçə dəfə seçilə bilər, hardakı, digərləri ümumiyyətlə seçilə bilməz. Əgər alqoritmin hər dövrdə hər bir dəyərdən istifadə etdiyindən əmin olmaq istəyirsinizsə, bir yanaşma kimi, öyrənmə dəstini qarışdırmaq olar (giriş xüsusiyyətlərini və labelları birlikdə qarışdırğınızdan əmin olun), sonra dəyərlərin üstündən bir bir keçmək, sonra yenə qarışdırmaq və bu şəkildə davam etmək. Ancaq, bu yanaşma daha mürəkkəbdir və ümumiyyətlə nəticəni yaxşılaşdırmır.



Şəkil 4-10. Stoxastik qradient enişin ilk 20 addımı

XƏBƏRDARLIQ

Stoxastik qradient enişindən istifadə edərkən, orta hesabla parametrlərin qlobal optimala doğru çəkilməsini təmin etmək üçün, dəyərlər müstəqil və eyni şəkildə paylanmış (IID - independent and identically distributed) olmalıdır. Bunu təmin etməyin sadə yolu, öyrənmə zamanı dəyərləri qarışdırmaqdır (məsələn, hər bir dəyəri ixtiyari şəkildə götürmək və ya hər dövrün əvvəlində öyrənmə dəstini qarışdırmaq). Dəyərləri qarışdırmasanız, məsələn, labellar üzrə sıralasanız, o zaman, stoxastik qradient eniş (SQE) hər bir labelı, bir bir optimallaşdırmağa başlayacaq və qlobal minimuma yaxınlaşa bilməyəcək.

—-

Scikit-Learn ilə stoxastik qradient enişi (SQE) istifadə edərək xətti reqresiyyanı yerinə yetirmək üçün, siz SGDRegressor sinifi istifadə edə bilərsiniz, hansı ki, susma halına görə orta kvadrat xətasının xərc funksiyasını optimizallaşdırır. Sıradaki kod, maksimum 1,000 epoach qədər (max_iter) və ya 100 (n_iter_no_change) epoach (dövr) ərzində itki 10^{-5}-dən (tol) aşağı düşənə qədər icra olunur. O, susma halına görə olan öyrənmə cədvəlindən (istifadə etdiyimizdən fərqlidir) istifadə edərək, 0.01 (eta0) öyrənmə dərəcəsi ilə başlayır. Son olaraq, bu, heç bir tənzimləmədən istifadə etmir (penalty=None; ətraflı məlumat birazdan):

```
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor(max_iter=1000, tol=1e-5, penalty=None, eta0=0.01,
                       n_iter_no_change=100, random_state=42)
sgd_reg.fit(X, y.ravel())  # y.ravel() because fit() expects 1D targets
```

Bir daha, normal tənliyin qaytardığı nəticəyə çox yaxın bir nəticə eldə etmisiniz:

```
>>> sgd_reg.intercept_, sgd_reg.coef_
(array([4.21278812]), array([2.77270267]))
```

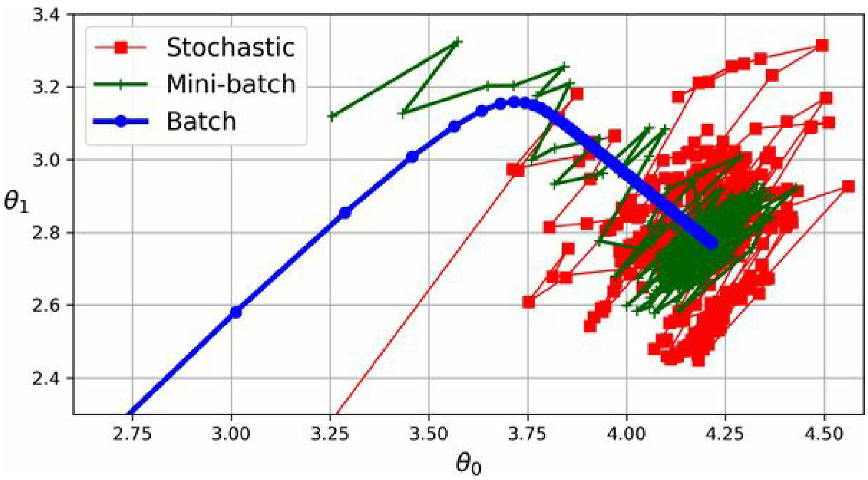
İPUCU

Bütün Scikit-Learn təxminçilər (estimators) `fit()` metoddan istifadə edərək təlimləndirilə bilər, amma bəzi təxminçilər (estimators) `partial_fit()` metoduna sahibdilər, hansını ki, siz tək bir təlim dövründə bir və ya bir neçə dəyişən üzərində icra etmək üçün çağıra bilərsiniz (o, `max_iter` və ya `tol` kimi hiperparametrləri iqnore edir). `partial_fit()` metodunu təkrar təkrar çağırmaq, modeli tədricən təlimləndirəcək. Təlim prosesində daha çox nəzarət lazım olduqda, bu yol faydalıdır. Digər modellər `warm_start` hiperparametrinə sahibdirlər (və bəziləri hər ikisinə); əgər siz, `warm_start=True` kimi təyin etsəniz, `fit()` methodun təlim olmuş modelin üzərində icrası modeli sıfırlamayacaq; o, `max_iter` və `tol` kimi hiperparametrlərə riayət edərək, təlimlərə qaldığı yerdən davam edəcək. Qeyd edək ki, `fit()` methodu öyrənmə cədvəli tərəfindən istifadə edilən iterasiya sayğacını (counter) sıfırlayır, lakin `partial_fit()` bunu etmir.

Mini-Toplu Qradient Eniş

Baxacağımız sonuncu qradient eniş alqoritmi “mini-toplu qradient eniş” alqoritmi adlanır. Toplu və qradient eniş alqortmi bildikdən sonra, bu alqoritm çox sadədir: hər addımda qradientləri, tam təlim dəstinə əsasən (TQE-də olduğu kimi) və ya bir dəyişənə (SQE-də olduğu kimi) əsasən hesabalamaq əvəzinə, mini-toplu qradient eniş alqoritmi qradientləri, “mini-dəstlər” adlanan kiçik ixtiyari dəstlər əsasında hesablayır. “MTQE” alqoritmin SQE-dən əsas üstünlüyü ondan ibarətdir ki, matris əməliyyatlarının hardware optimallaşdırılmasından performans artımı əldə edə bilərsiniz, xüsusiylə GPU-lardan istifadə etdikdə.

Parametrlər fəzasında, alqoritmin progresi SQE alqoritmi ilə müqayisədə daha az nizamsızdır, xüsusən də kifayət qədər böyük “mini-dəstlər”də. Nəticə olaraq, mini-toplu QE (qradient eniş), stoxastik QE ilə müqayisədə minimuma daha yaxın ətrafda hərəkət edcək, amma onun daxili minimumdan qaçması daha çətin ola bilər (orta kvadrat xətası dəyər funksiyası ilə xətti reqresiyadan fərqli olaraq daxili minimumdan əziyyət çəkən problemlər halında). Şəkil 4-11, üç qradient eniş alqoritmin təlim zamanı parametr fəzasindaki çəkdiyi yolun təsvir edir. Onların hanısı minimuma yaxın yerləşir, amma toplu QE isə minimumda dayanır, halbuki həm stoxastik QE və mini-toplu QE ətrafda hərəkət etməyə dava edir. Bununla belə, unutmayın ki, toplu QE hər bir addım üçün daha çox vaxt sərf edir, və yaxşı təlim cədvəlindən istifadə etdiyiniz halda, stoxastik QE və mini-toplu QE-də minimuma çatacaqlar.



Şəkil 4-11. Parametr fəzasında qradient enişin yolları

Cədvəl 4-1 xətti reqresiya üçün müzakirə etdiyiniz alqoritmləri müqayisə edir (*m* təlim dəyişənlərinin sayıdır, *n* isə xüsusiyyətlərin sayı).

Alqoritm	Böyük m	Out-of-Core Support	Böyük n	Hiper parametrlər	Scaling tələb olunur	**Scikit-Learn**
Normal tənlik	Sürətli	Yox	Yavaş	0	Yox	N/A
SVD	Sürətli	Yox	Yavaş	0	Yox	Xətti Reqressiya
Toplu QE	Yavaş	Yox	Sürətli	2	Bəli	N/A
Stoxastik QE	Sürətli	Bəli	Sürətli	≥ 2	Bəli	SGDRegressor
Mini-Toplu QE	Sürətli	Bəli	Sürətli	≥ 2	Sürətli	N/A

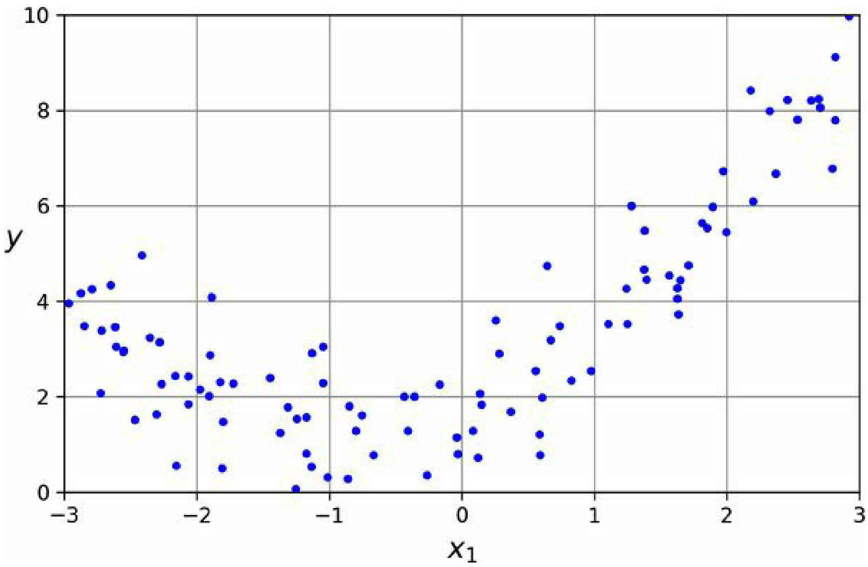
Təlimdən sonra demək olar ki, heç bir fərq yoxdur: bütün alqoritmlər çox oxşar modellərlə neticələnir və eyni şəkildə proqnozlar verirlər.

Polinom Reqressiya

Birdən sizin datanız düz xətti yox, daha mürəkkəbdirse? Qəribədir ki, qeyi-xətti datanı uyğunlaşdırmaq üçün xətti modeldən istifadə edə bilərsiniz. Ən sadə yolu, hər bir xüsusiyyətin qüvvəsini yeni xüsusiyyət kimi artırmaq, sonra isə, bu genişləndirilmiş xüsusiyyətlər toplusu üzərində xətti modeli təlimləndirmək. Bu yöntəm, "polinom reqressiya" (polynomial regression) adlanır.

Bir nümunəyə baxaq. Birincisi, biz, sadə kvadrat tənliyinə (yəni, $y = ax^2 + bx + c$) və əlavə biraz "noice"-a əsaslanaraq müəyyən qeyri-xətti data yaradacağıq:

```
np.random.seed(42)
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X ** 2 + X + 2 + np.random.randn(m, 1)
```



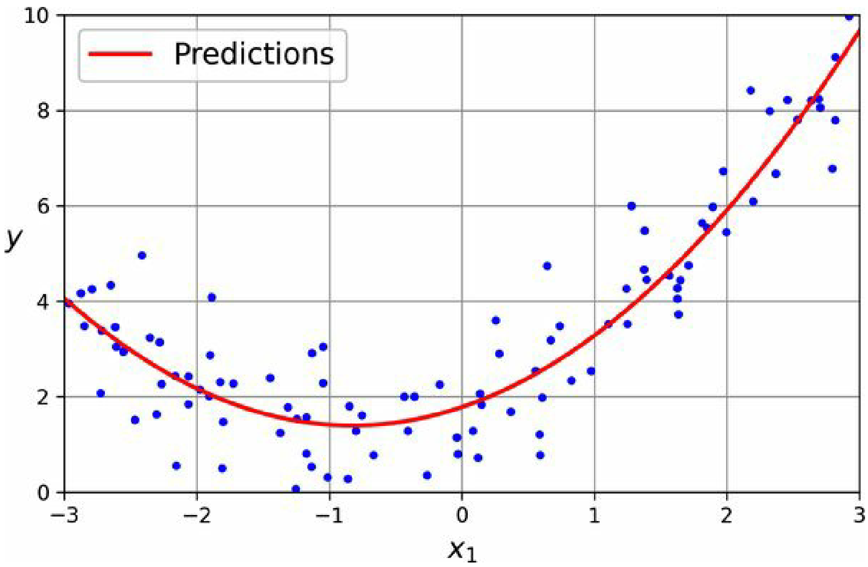
Şəkil 4-12. Əmələ gətirilmiş qeyri-xətti və "noisy" data toplusu

Aydındır ki, düz xətt heç vaxt bu data-ya düzgün şəkildə uyğunlaşmacaq. Onda gəlin bizim təlim məlumatlarımızı dəyişdirmək üçün Scikit-Learn-nın PolynomialFeatures sinifindən istifadə edək, təlim toplusunda olan hər bir xüsusiyyətin kvadratını (ikinci dərəcəli polinom), yeni xüsusiyyət kimi əlavə edək (bu halda yanlız bir xüsusiyyət var):

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly_features = PolynomialFeatures(degree=2, include_bias=False)
>>> X_poly = poly_features.fit_transform(X)
>>> X[0]
array([-0.75275929])
>>> X_poly[0]
array([-0.75275929,  0.56664654])
```

İndi, "X_poly" özündə X-in bütün əsl (orijinal) xüsusiyyətləri və əlavə olunmuş xüsusiyyətlərin kvadratlarnı özündə saxlıyır. İndi biz genişləndirilmiş təlim məlumatlarını Xətti Reqressiya (LinearRegression) modelinə uyğunlaşdıra bilərik (Şəkil 4-13):

```
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X_poly, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([1.78134581]), array([[0.93366893, 0.56456263]]))
```



Şəkil 4-13. Polinom reqressiya modelinin proqnozu

Pis deyil, Orijinal funksiya $y = 0.5{x_1}^2 + 1.0x_1 + 2.0 + Gaussian\ noise$ olduğu halda, model $\hat{y} = 0.56{x_1}^2 + 0.93x_1 + 1.78$ kimi dəyərləndirir.

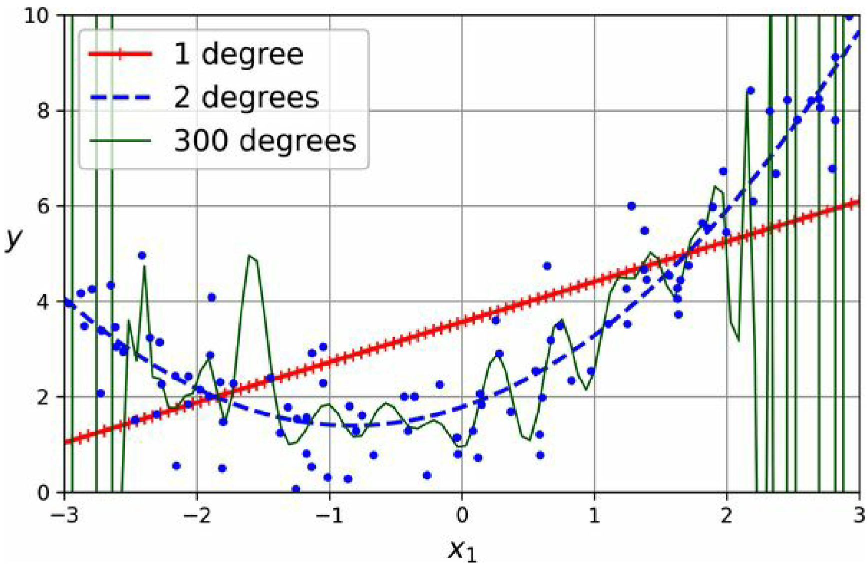
Qeyd edək ki, burada bir neçə xüsusiyyət olduqda, polinom reqressiya modeli xüsusiyyətlər arasında olan əlaqələri tapmağa qadirdir, bu isə, düz xətti reqressiya modelinin edə bilməcəyi bir şeydir. Bu, PolynomialFeatures-in verilən dərəcəyə qədər bütün xüsusiyyətlərin kombinasiyaları əlavə etməsi ilə mümkün olur. Misal üçün, iki a və b xüsusiyyətləri olsaydı, $degree = 3$ olan PolynomialFeatures, yalnız $a2, a3, b2, b3$ yox, həmçinin ab, a^2b, ab^2 kimi birləşmələri də əlavə edərdi.

XƏBƏRDARLIQ

`PolynomialFeatures(degree= d )`, n xüsusiyyətdən ibarət massivi, $(n + d)\ /\ d!n!$ xüsusiyyəti olan massivə çevirir. Burada, n!, n-nin faktoriyalı deməkdir və $1 \times 2 \times 3 \times ... \times n$ ifadəsinə bərabərdir. Xüsusiyyətlərin sayısının şişib partlamasından ehtiyatlı olun.

Öyrənmə Əyriləri

Yüksək dərəcəli polinom reqresiyyanı həyata keçirsəz, çox güman ki, təlim datasına düz xətti reqresiyyadan daha yaxşı uyğunlaşacaqsınız. Misal üçün, Şəkil 4-14 təlim datasına (əvvəl istifadə etdiyimiz, yuxarı bax) 300 dərəcəli polinom modelini tətbiq edir və nəticəni təmiz (pure) düz xətt modeli və kvadart modeli (ikinci dərəcəli polinom) ilə müqayisə edir. Diqqət yetirin, 300 dərəcəli polinom modeli təlim nümunələrinə mümkün qədər yaxın ola bilmək üçün necə oyana-buyana hərəkət edir.



Şəkil 4-14. Yüksək dərəcəli polinom reqresiyya

Bu çox dərəcəli polinom reqresiyya modeli təlim datasına ciddi şəkildə həddindən çox uyğunlaşır (overfitting), hardakı, düz xətti modeli isə ona uyğunlaşmır (underfitting). Bu halda ən yaxşı uyğunlaşan model kvadrat modeldir, çünki məlumat (data) kvardart modeldən istifadə edilərək yaradılıb. Ancaq ümumiyyətlə, siz məlumatı (datanı) hansı funksiya yaratdığını bilməyəcəksiniz, bu halda, modelinizin nə dərəcədə mürəkkəb olduğunu necə qərar verə biləcəksiniz? Modelinizin dataya həddən artıq uyğun və ya uyğun olmadığını necə deyə bilərsiniz?

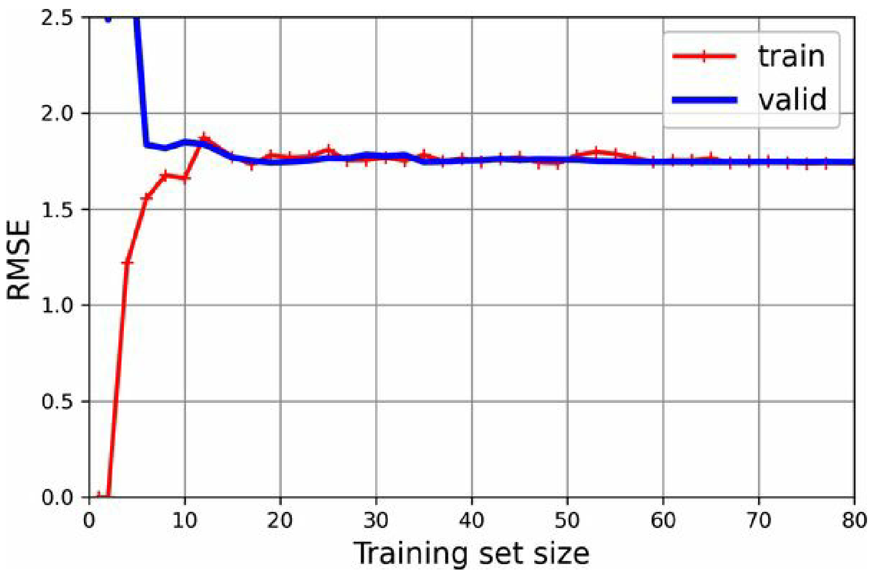
Fəsil 2də, siz modelin ümumiləşdirmə performasını qiymətləndirmək üçün çarpaz-doğrulamadan istifadə etmişdiniz. Əgər model təlim datası üzərində yaxşı işləyirsə, lakin çarpaz-doğrulama göstəricilərinə görə zəif ümumiləşirsə, o zaman modeliniz həddən artıq uyğunlaşır deməkdir. Hər ikisində də zəif işləyirsə, deməli uyğunlaşmır. Bu, modelin çox sadə və y çox mürəkkəb olduğunu müəyyən etməyin bir yoludur.

Başqa bir yol isə, öyrənmə əyrisinə baxmaq, hansı ki, təlim iterasiyasının funksiyası kimi modelin təlim xətasını və validasiya (validation) xətasını təsvir edir: sadəcə olaraq, həm təlim dəsti, həm də doğrulama dəsti üzrə təlim zamanı müntəzəm fasilələrlə modeli qiymətləndirin, və nəticələri tərtib edin. Əgər modeli tədricən təlimləndirmək mümkün olmursa (yəni, `partial_fit()` və `warm_start` dəstəklənirsə), o zaman siz onu təlim dəstini tədricən daha böyük alt çoxluqlarında bir neçə dəfə təlimləndirməlisiniz.

Scikit-Learn, bu işdə çox faydalı olacaq `learning_curve()` funksiyasına sahibdir: o, çarpaz-doğrulamadan istifadə edərək modeli təlimləndirir və qiymətləndirir. Susma halına görə (by default), o, modeli təlim dəstinin artan alt çoxluqları üzrə yenidən hazırlayır, amma əgər model artımlı öyrənməyi dəstəkləyirsə, o zaman siz, `learning_curve()` funksiyasını çağıran zaman `exploit_incremental_learning=True` təyin edə bilərsiniz və bu da modeli tədricən təlimləndirməyə imkan yaradır. Funksiya həm modeli qiymətləndirdiyi təlim dəstinin ölçüsünü, həm də hər ölçü və hər çarpaz-doğrulama qatı üçün ölçdüyü təlim və yoxlama ballarını qaytarır. Düz xətti reqresiyya modelinin öyrənmə əyrilərinə baxmaq üçün bu funksiyadan istifadə edək (Şəkil 4-15).

```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, valid_scores = learning_curve(
    LinearRegression(), X, y, train_sizes=np.linspace(0.01, 1.0, 40), cv=5,
    scoring="neg_root_mean_squared_error")
```

```
train_errors = -train_scores.mean(axis=1)
valid_errors = -valid_scores.mean(axis=1)

plt.plot(train_sizes, train_errors, "r-+", linewidth=2, label="train")
plt.plot(train_sizes, valid_errors, "b-", linewidth=3, label="valid")
[...]  # beautify the figure: add labels, axis, grid, and legend
plt.show()
```



Şəkil 4-15. Öyrənmə əyriləri

Bu model uyğunlaşmır (underfitting). Səbəbini öyrənmək üçün əvvəlcə təlim xətalarına baxaq. Təlim çoxlusunda yalnız bir və yay iki nümunə olduqda, model onlara mükəmməl uyğunlaşa bilir, buna görə də əyri sıfırdan başlayır. Lakin təlim çoxluğuna yeni nümunə əlavə olunduqca modelin təlim çoxluğuna mükəmməl uyğunlaşması qeyri-mümkün olur, çünki data səs-küylüdür və ümumiyyətlə xətti deyildir. Beləliklə, təlim datasında xəta bir nöqtəyə çatana kimi yüksəlir, o nöqtəyə ki, təlim dəstinə yeni nümunənin əlavə olunması, ortalama xətanı daha yaxşı və ya daha pis etmir. İndi isə validasiya xətasına baxaq. Model çox az nümunələri üzərində təlimləndirilibsə, onda o, düzgün ümumiləşməyə qadir deyil, buna görə də validasiya xətası əvvəldə olduqca böyükdür. Sonra, modelə daha çox təlim nümunələri göstərdikcə, o öyrənir ve beləliklə, validasiya xətası getdikcə azalmağa başlıyır. Bununla belə, bir daha düz xətt datanı yaxşı modelləşdirə bilmir, buna görə də xəta yaylada bitir, hansı ki digər əyriyə çox yaxındır.

Bu öyrənmə əyriləri uyğunlaşmayan modellər üçün xarakterikdir. Hər iki əyri yaylaya çatır; onların hər ikisi kifayət qədər yaxın və yüksəkdir.

—-

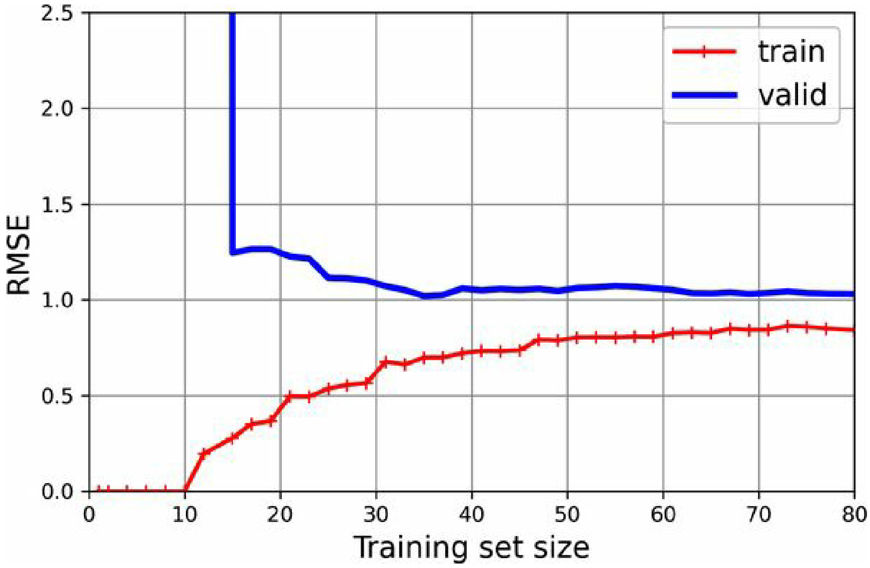
İPUCU

Modeliniz təlim datasına uyğunlaşmırsa, daha çox nümunələrin əlavə etmək kömək etməyəcək. Siz daha yaxşı bir model istifadə etməli və ya daha yaxşı xüsusiyyətlərə sahib olmalısınız.

—-

İndi isə, eyni data üzərində 10-cu dərəcəli polinom modeli üçün öyrənmə əyrilərinə baxaq (Şəkil 4-16):

```
from sklearn.pipeline import make_pipeline

polynomial_regression = make_pipeline(
    PolynomialFeatures(degree=10, include_bias=False),
    LinearRegression())

train_sizes, train_scores, valid_scores = learning_curve(
    polynomial_regression, X, y, train_sizes=np.linspace(0.01, 1.0, 40), cv=5,
    scoring="neg_root_mean_squared_error")
[...]  # same as earlier
```



Şəkil 4-16. 10-cu dərəcəli polinom modeli üçün öyrənmə əyriləri

Bu öyrənmə əyriləri əvvəl gördüyümüz əyrilərə benzəyir, lakin onlar arasında çox mühüm iki fərq var:

- Təlim datasında xəta əvvəlkindən xeyli aşağıdır.
- Əyrilər arasında boşluq var. Bu o deməkdir ki, model validasiya datasından daha çox təlim datası ilə daha yaxşı işləyir, hansı ki, uyğunlaşan modelin əlamətləridir. Amma əgər daha böyük bir öyrənmə çoxluğunu istifadə etsəniz, iki əyri bir birinə yaxınlaşmağa davam edəcəkdi.

—-

İPUCU

Həddindən artıq uyğunlaşan modeli təkmilləşdirməyin bir yolu, validasiya xətasını öyrənmə xətasına çatana qədərona daha çox öyrənmə datasını verməkdir.

—-

QƏRƏZ/FƏRQLİLİK MÜBADİLƏSİ

—-

Statistikanın və maşın öyrənməsinin mühüm nəzəri nəticəsi olan modelin ümumiləşdirmə xətasını, üç çox fərqli səhvin cəmi kimi ifadə etmək olar:

Qərəz (Bias)

Ümumilləşdirmə xətasının bu hissəsi yanlış fərziyyələrdən qaynaqlanır, məsələn, verilənlərin əslində kvadratik olduğu halda xətti olduğunu fərz etmək. Yüksək qərəzli (high-bias) model, çox güman ki, təlim datasına uyğun gəlmir.

Fərqlilik (Variance)

Bu hissə modelin təlim datasında olan hər hansı kiçik dəyişikliklərə, həddindən artıq hessaslığı ilə bağlıdır. Çoxsaylı sərbəstlik dərəcələrinə malik model (məsələn, yüksək dərəcəli polinom model) çox güman ki, yüksək fəqrliliyə sahibdir və beləliklə, təlim datasına uyğunlaşır.

Azalmaz xəta (Irreducible error)

Bu hissə məlumatların özünün səs-küylü olması ilə əlaqədardır. Xətanın bu hissəsini azaltmağın yeganə yolu məlumatları təmizləməkdir (məsələn, sıradan çıxmış sensorların məlumatlarını düzəltmək və ya kənar göstəriciləri aşkar edib aradan qaldırmaq).

Modelin mürəkkəbliyinin artırılması adətən onun fərqliliyini (varience) artıracaq amma qərəzliliyini azaldacaq. Əksinə, modelin mürükkəbliyinin azalması onun qərəzliliyini (bias) artıracaq amma fərqliliyini azaldacaq. Buna görə də buna mübadilə deyilir.

Müntəzəmləşdirilmiş (Nizamlanmış) xətti modellər

1-ci və 2-ci Fəsillərdə gördüyümüz kimi, həddən artıq uyğunlaşmanı azatlmağın yaxşı yolu modeli nizamlamaqdır (yəni, onu məhdudlaşdırmaqdır): onun sərbəstlik dərəcəsi nə qədər az olarsa, məlumatı üstələmək bir o qədər çətin olacaq. Polinom modelini nizamlamağın sadə yolu polinom dərəcələrin sayını azatmaqdır.

Xətti model üçün nizamlanma adətən modelin çəkilərini məhdudlşadırmaqla əldə edilir. İndi biz çəkiləri məhdudlaşdırmaq üçün üç fərqli üsul tətbiq edən reqressiyalara (silsilə (ridge) reqresiyyası, lasso reqressiyası, elastik xalis (elastic net) reqresiyyası) baxacağıq.

Silsilə Reqresiyyası

Silsilə reqresiyyası (Ridge Regression), həmçinin Tixonov nizamlanması adlanır, xətti reqresiyyasının nizamlanmış versiyasıdır: nizamlanma dövrü bərabərdir,

$$\frac{\alpha}{m}\sum_{i=1}^{n}\theta_i^2$$

hansı ki, orta kvadrat xətasına (OKX) əlavə olunur. Bu, öyrənmə alqoritmini təkcə verilənlərə uyğunlaşmamağa, həm də modelin çəkilərini mümkün qədər kiçik saxlamağa məcbur edir. Qey edək ki, nizamlanma dövrü yanlız təlim zamanı xərc funksiyasına əlavə edilməlidir. Model təlimləndirdikdən sonra, siz modelin performansını qiymətləndirmək üçün nizamlanmamış orta kvadrat xətasından (OKX, vəya kök orta kvadrat xətasından (KOKX)) istifadə edə bilərsiniz.

Hiperparameter α, modeli nə qədər nizamlamaq istədiyinizi idarə edir. Əgər $\alpha = 0$ olarsa, o zaman silsilə reqresiyyası sadəcə xətti reqresiyyadır. Əgər α çox böyükdürsə, onda bütün çəkilər sıfıra çox yaxın olacaq və nəticə olaraq düz xətt olacaq, hansı ki, məlumat ortalamasından keçir. Tənlik 4-8 silsilə reqresiyyasının xərc funksiyasını təqdim edir.

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \frac{\alpha}{m}\sum_{i=1}^{n}{\theta_i}^2$$

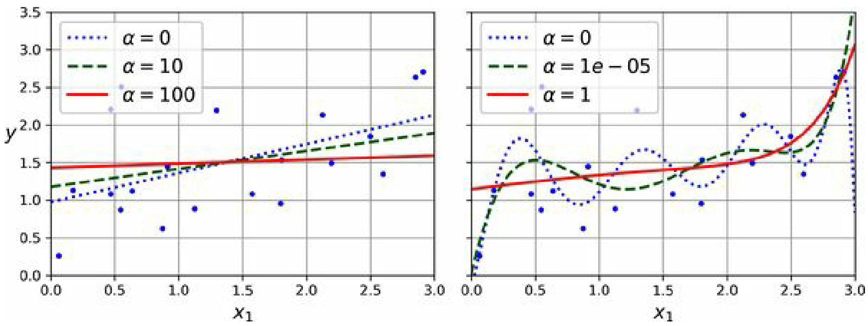
Tənlik 4-8. Silsilə reqresiyyasının xərc funksiyası

Qeyd edək ki, θ_0 qərəzli ifadəsi nizamlanmamışdır ($i = 1$-dən başlıyır, 0-dan yox). Əgər biz, $\mathbf{w}$-ni xüsusiyyət çəkilərinin vektoru kimi təyin etsək (θ_0-θ_n), o zaman nizamlanma ifadəsi $\alpha\left(\|\mathbf{w}\|_2\right)2/m$ ifadəsinə bərabər olacaq. Burada $\|\mathbf{w}\|_2$, ℓ_2 çəki vektorunun normasını təmsil edir. Toplu qradient enişi üçün qərəzli ifadəsinin qradientinə heç nə əlavə etmədən, orta kvadrat xətasının (OKX) qradient vektorunun xüsusiyyət çəkilərinə uyğun gələn hissəsinə sadəcə $2\alpha\mathbf{w} \ / \ m$ əlavə edin (Tənlik 4-6).

XƏBƏRDARLIQ

Silsilənin reqressiyasını həyata keçirməzdən əvvəl məlumatların miqyasını artırmaq vacibdir (məsələn, StandardScaler istifadə etməklə), çünki o, giriş xüsusiyyətlərinin miqyasına həssasdır. Bu, nizamlı modellərin əksəriyyətinə aiddir.

Şəkil 4-17, müxtəlif α dəyərlərindən istifadə edərək bəzi çox səs-küylü xətti məlumatlarla təlimləndirilmiş bir neçə silsilə modelini göstərir. Solda düz silsilə modeli istifadə olunur, bu da xətti proqnoza səbəb olur. Sağda, məlumat əvvəlcə `PolynomialFeatures(degree=10)` istifadə edərək genişləndirilir, sonra `StandardScaler` istifadə edərək miqyası böyüdülür, və nəhayət, əldə edilən xüsusiyyətlərə silsilə modeli tətbiq olunur: bu da, silsilə nizamlanması ilə polinom reqressiyadır. Qeyd edək ki, α-nın artırılması daha düz proqnozlara (yəni, daha az ekstremal, daha ağlabatan) gətirib çıxarır, beləcə, modelin fərqliliyi azaldır, lakin onun qərəzliliyini artırır.



Şəkil 4-17. Xətto (solda) və polinom (sağda) modellər, müxtəlif səviyyəli silsilə nizamlanma ilə

Xətti reqressiyada olduğu kimi, biz ya “yaxın-forma” (closed-form) tənliyi hesablamaqla, ya da qradient enişi yerinə yetirməklə silsilə reqressiyasını həyata keçirə bilərik. Müsbət və mənfi cəhətləri eynidir. Tənlik 4-9, “yaxın-forma” tənliyinin həllini göstərir, burada $\mathbf{A}$, $(n+1)\times(n+1)$ ilə ifadə olan eynilik matrisidir və qərəz terminə uyğundur (yuxarı sol xanada 0-dan başqa).

$$\widehat{\boldsymbol{\theta}} = \left(\mathbf{X}^\top\mathbf{X} + \alpha\mathbf{A}\right)^{-1}\mathbf{X}^\top\mathbf{y}$$

Tənlik 4-9. Silsilə reqresiyyasının ”yaxın-forma” həlli

Burada, Scikit-Learn ilə "yaxın-forma" həllini istifadə edərək silsilə reqressiyasını necə yerinə yetirmək olar göstərilir (Tənlik 4-9.-un başqa bir variantı, hansı ki, André-Louis Cholesky-nin matris faktorizasiya texnikasından istifadə edir).

```
>>> from sklearn.linear_model import Ridge
>>> ridge_reg = Ridge(alpha=0.1, solver="cholesky")
>>> ridge_reg.fit(X, y)
>>> ridge_reg.predict([[1.5]])
array([[1.55325833]])
```

Və stoxastik qradient enişi istifadə edərək göstərilir.

```
>>> sgd_reg = SGDRegressor(penalty="l2", alpha=0.1 / m, tol=None,
...                         max_iter=1000, eta0=0.01, random_state=42)
...
>>> sgd_reg.fit(X, y.ravel())  # y.ravel() because fit() expects 1D targets
>>> sgd_reg.predict([[1.5]])
array([1.55302613])
```

İstifadə ediləcək nizamlanma termini cərimə (penalty) hiperparametri tərəfindən təyin edilir. "l2" onu göstərir ki, siz SQE-i (Stoxastik Qradient Enişi) OKX-nın (orta kvadrat xətası) dəyər funksiyasına çəki vektorunun ℓ_2 normasının kvadratının alfa çarpımına bərabər nizamlanma ifadəsini əlavə etməsini istəyirsiniz. Bu silsilə reqressiyasına bənzəyir, yanlız bu halda *m*-ə bölmə yoxdur; buna görə də, `Ridge(alpha=0.1)` ilə eyni nəticə almaq üçün, `alpha=0.1/m` kimi təyin etdik.

İPUCU

—

`RidgeCV` sinifi də həmçinin silsilə reqresiyyasını həyata keçirir, lakin o, çarpaz-doğrulamadan istifadə edərək avtomatik olaraq hiperparametrləri sazlayır. Bu, təqribən `GridSearchCV` sinifindən istifadəyə bərabərdir, lakin o, silsilə reqresiyyası üçün optimallaşdırılıb və daha sürətli işləyir. Bir neçə digər texmin mexanizmləri (əsasən xətti) də effektiv CV variantlarına sahibdirlər, məsələn `LassoCV` və ya `ElasticNetCV`.

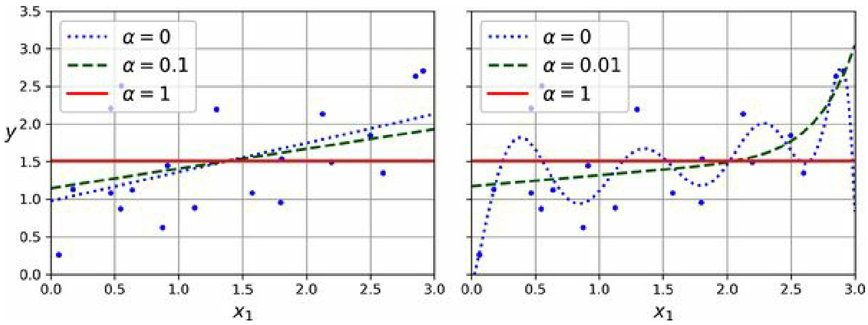
Lasso Reqresiyyası

Sadə olaraq Lasso Reqressiyası adlanan reqressiya, xətti reqressiyanın başqa bir nizanlanmış versiyasıdır. Açılışı isə, "*Ən Az Mütləq Daralma və Seçim Operatoru Reqressiyası*"-dır (Least Absolute Shrinkage and Selection Operator Regression). Silsilə reqressiyası kimi, bu reqressiyada dəyər funksiyasına tənzimləmə ifadəsini əlavə edir. Lakin o, ℓ_2 normasının kvadratı əvəzinə çəki vektorunun ℓ_1 normasından istifadə edir (Tənlik 4-10). Diqqət yetirin ki, silsilə reqresiyyasında ℓ_2 norması 2α ilə vurulur, halbuki ℓ_2 norması $\alpha \ / \ m$ ilə vurulur. Bu amillər, optimail α dəyərinin təlim çoxluğunun ölçüsündən asılı olmadığını təmin etmək üçün seçilmişdir: müxtəlif normalar müxtəlif amillərə səbəb olur (ətraflı məlumat üçün, Scikit-Learn buraxış #15657-yə baxın).

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + 2\alpha \sum_{i-1}^{n} \theta_i$$

Tənlik 4-10. Lasso reqressiyasının dəyər funksiyası

Şəkil 4-18, əvvəlki şəkildəki kimi (Şəkil 4-7) eyni şeyi göstərir, lakin silsilə modeli əvəzinə lasso modelini və fərqli α dəyərini istifadə edir.

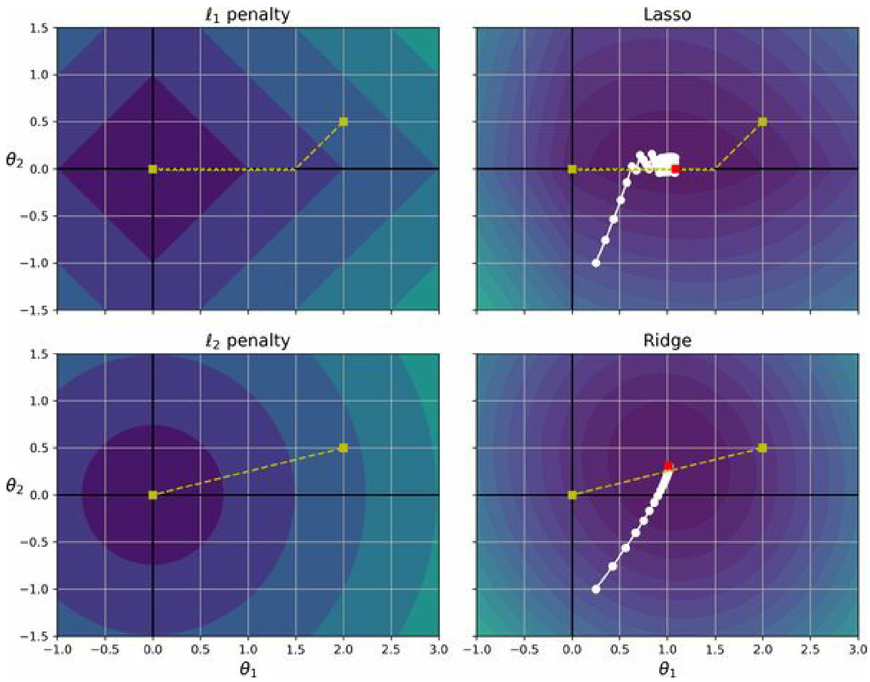


Şəkil 4-18. Xətti (sol) və polinom (sağ) modellər, hər ikisi müxtəlif səviyyəli lasso reqresiyyasından istifadə edir

Lasso reqressiyasının mühüm xüsusiyyəti ondan ibarətdir ki, o, ən az vacib xüsusiyyətlərin çəkilərini aradan qaldırmağa meyillidir (yəni, onları sıfıra mənimsədir). Məsələn, Şəkil 4-18-də sağ tərəfdəki (α = 0.01) kəsikli xətt təqribən kub şəklində görünür: yüksək dərəcəli polinom xüsusiyyətlər üçün bütün çəkilər sıfıra bərabərdir. Başqa sözlə, lasso reqressiyası, funksiya seçimini özü avtomatik yerinə yetirir və sıfırdan fərqli bir neçə xüsusiyyət çəkiləri olan seyrək (sparse) modelini əmələ gətirir.

Şəkil 4-19-a baxaraq bunun niyə belə olduğunu başa düşə bilərsiniz: Oxlar iki model parametrlərini, fon konturları isə müxtəlif dəyər funskiyasılarını təmsil edir. Sol yuxarı hissədə, konturlar istənilən oxa yaxınlaşdıqca xətti şəkildə azalan ℓ_1-nin $(|\theta_1| + |\theta_2|)$ itkisini təmsil edir. Məsələn, əgər siz model parametrlərini $\theta_1 = 2$ və $\theta_2 = 0.5$ olaraq təyin etsəniz, qradient eniş hər iki parametri bərabər şəkildə azaldacaq (kəsikli sarı xətt ilə göstərildiyi kimi); buna görə də birinci θ_2 sıfıra çatacaq (çünki o, başlanğıcda sıfıra yaxın idi). Bundan sonra, qradient eniş $\theta_1 = 0$ olana qədər aşağı düşəcək (ətrafda biraz hoppanmaqla, çünki ℓ_1 qradienti heç vaxt sıfıra yaxınlaşmır, hər bir parametr üçün ya -1 ya da 1 olur).

Yuxarı sağ hissədə, konturlar lasso reqressiyasının dəyər funksiyasını təyin edir (yəni, orta kvadrat xətasının dəyər funksiyası üstəgəl ℓ_1-nin itkisi). Kiçik ağ dairələr, $\theta_1 = 0.25$ və $\theta_2 = -1$ ətrafında inisializasiya olunmuş bəzi model parametrlərini optimallaşdırmaq üçün qradient enişinin keçdiyi yolu göstərir: yolun, necə sürətli şəkildə $\theta_2 = 0$-a çatdığına bir daha diqqət yetirin, sonra sıxlıqdan aşağı yuvarlanır və qlobal optimum ətrafında hoppanır (qırızı kvadrat ilə göstərilib). Əgər biz α-nın dəyərini artırsaq, qlobal optimum kəsikli sarı xətt boyunca sola, yox əgər α-nın dəyərini azaltsaq, qlobal optimum sağa hərəkət edəcək (bu misalda, nizamlanmamış OKX (orta kvadrat xətası) üçün optimal parametrlər $\theta_1 = 2$ və $\theta_2 = 0.5$ şəklində təyin olunub).



Şəkil 4-19. Lasso və Silsilə Reqressiyası

Aşağıdakı iki hissə eyni şeyi göstərir, lakin ℓ_2 penaltisi ilə. Sol alt hissədə, mənbəyə yaxınlaşdıqca ℓ_2-i itkisinin azaldığını görə bilərsiniz, ona görə də qradient eniş həmin nöqtəyə qədər düz bir yol tutur. Aşağı sağ hissədə isə, konturlar silsilə reqresiyyasının dəyər funksiyasını təmsil edir (yəni, OKX (orta kvadrat xətasının dəyər funksiyası üstəgəl ℓ_2 itkisi). Gördüyünüz kimi, parametrlər qlobal optimuma yaxınlaşdıqca qradientlər kiçilməyə başlıyır, öna görə də qradient eniş təbii olaraq yavaşlamağa başlıyır. Bu, ətrafdakı hoppanmağı məhdudlaşdırır, hansı ki, silsilə reqresiyyasına lasso reqresiyyasından daha sürətli birləşməsinə kömək edir. Həmçinin qeyd edək ki, optimal parametrlər (qırmızı kvadrat ilə təsvir olunub) α-nın dəyərini artırdığınız zaman mənbəyə daha da yaxınlaşır, lakin onlar heç vaxt tamamilə aradan qalmırlar.

İPUCU

—-

Lasso reqressiyasından istifadə edərkən, sonda qradient enişin optimum ətrafında hoppanmaması üçün təlim zamanı öyrənmə sürətini tədricən azaltmalısınız. O, optimum ətrafında hələ də hoppanacaq, lakin addımlar gətdikcə daha da kiçik olacaq, ona görə də birləşəcək.

—-

Lasso reqressiyası $\theta_i = 0$ ($i = 1, 2, 3, ..., n$ üçün) ilə diferensiallaşmır, lakin $\theta_i = 0$ əvəzinə $\mathbf{g}$ subqradient vektorundan istifadə etsəniz, qradient eniş hələ də işləyəcək. Tənlik 4-11.-də, lasso dəyər funskiyası ilə qradient eniş üçün istifadə edə biləcəyiniz subqradient vektor tenliyini göstərir.

$$g(\theta, J) = \nabla_\theta \, \mathrm{MSE}(\theta) + 2\alpha \begin{pmatrix} \mathrm{sign}\,(\theta_1) \\ \mathrm{sign}\,(\theta_2) \\ \vdots \\ \mathrm{sign}\,(\theta_n) \end{pmatrix} \quad \text{where } \mathrm{sign}\,(\theta_i) = \begin{matrix} -1 & \text{if } \theta_i < 0 \\ 0 & \text{if } \theta_i = 0 \\ +1 & \text{if } \theta_i > 0 \end{matrix}$$

Tənlik 4-11. Lasso reqressiyasının subqradient vektoru

Aşağıdakı Scikit-Learn misalı, Lasso sinifindən istifadə edir:

```
>>> from sklearn.linear_model import Lasso
>>> lasso_reg = Lasso(alpha=0.1)
>>> lasso_reg.fit(X, y)
>>> lasso_reg.predict([[1.5]])
array([1.53788174])
```

Qeyd edək ki, bunun əvizinə siz `SGDRegressor(penalty="l1", alpha=0.1)` istifadə edə bilərsiniz.

Elastik Xalis Reqressiya

Elastik Xalis Reqressiya, silsilə reqressiya ilə lasso reqressiya arasında yerləşir. Nizamlanma termini həm silsilə, həm də lasso reqressiyasının nizamlanma terminlərinin çəki cəmidir və siz r qarışıq nisbətinə (mix ratio) nəzarət edə bilərsiniz. $r = 0$ olduqda EXR reqressiyası silsilə reqressiyasına, $r = 1$ olduqda isə lasso reqressiyasına bərabər olur (Tənlik 4-12).

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + r\left(2\alpha\sum_{i-1}^{n}\theta_i \mid\right) + (1-r)\left(\frac{\alpha}{m}\sum_{i-1}^{n}\theta_i^2\right)$$

Tənlik 4-12. Elastik Net Reqressiyasının dəyər funksiyası

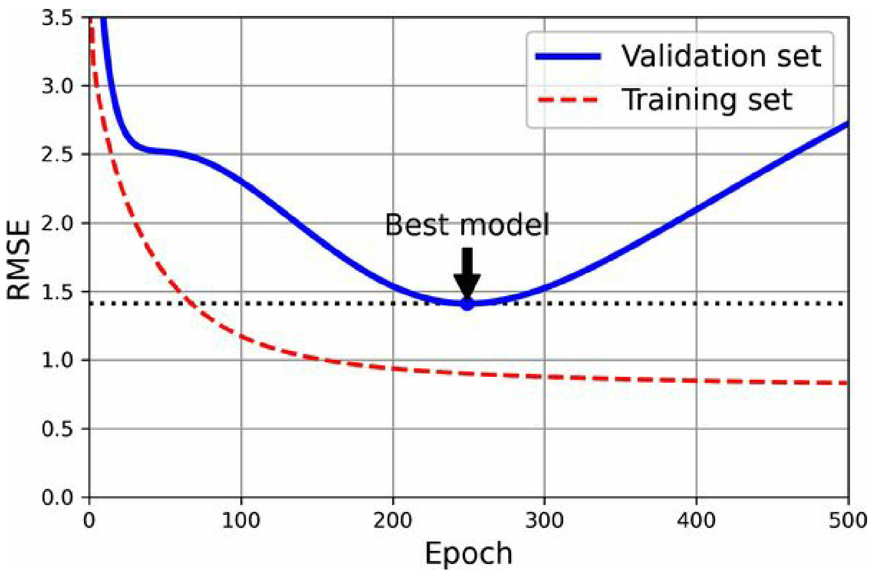
Beləliklə, nə vaxt hansı reqressiyadan istifadə etmək lazımdır (yəni, heç bir nizamlanmadan istifadə etmədən)? Demək olar ki, həmişə azda olsa nizamlanmaya üstünlük verilir, buna görə də ümumiyyətlə düz xətt reqressiyasından istifadə etməməlisiniz. Silsilə reqressiyası yaxşı seçimdir, lakin əgər düşünürsünüz ki yalnız bir neçə xüsusiyyətlər faydalıdır, o zaman lasso və ya elastik net reqressiyaya üstünlük verməlisiniz, çünki onlar (əvvəldə müzakirə etdiyimiz kimi) yararsız xüsusiyyətlərin çəkilərini sıfıra endirirlər. Ümumiyyətlə, elastik net reqressiyaya lasso reqressiyadan daha çox üstünlük verilir, çünki xüsusiyyətlərin sayı təlim nümunələrinin sayından çox olduqda və ya bir neçə xüsusiyyətlər güclü korrelyasiya olduqda lasso reqressiyası qeyri-sabit davrana bilər.

Scikit-Learn-nın `ElasticNet`-dən istifadə edən qısa bir nümunə (`l1_ratio`, r qarışıq nisbətinə uyğundur).

```
>>> from sklearn.linear_model import ElasticNet
>>> elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
>>> elastic_net.fit(X, y)
>>> elastic_net.predict([[1.5]])
array([1.54333232])
```

Tez Dayanma

İterativ öyrənmə alqoritmləri nizamlanmağın bir yolu (məsələn qradient eniş), validasiya xətasının minimuma çatan kimi təlimi dayandırmaqdır. Buna "tez dayanma" deyilir. Şəkil 4-20-də, əvvəl istifadə etdiyimiz kvadratik data çoxluğu əsasında toplu qradient eniş ilə öyrədilmiş mürəkkəb model əks olunur (bu halda, yüksək dərəcəli polinom reqresiyya modeli). Dövrlər (epoach) keçdikcə alqoritm öyrənir və onun təlim çoxluğundakı proqnoz xətası (KOKX - Kök Orta Kvadrat Xətası), validasiya çoxluğundakı proqnoz xətası ilə birlikdə aşağı düşür. Bir müddət sonra validasiya xətası azalmağı dayandırır və yenidən yüksəlməyə başlıyır. Bu, modelin təlim çoxluğuna uyğunlaşmağa başlanğıcını göstərir. Tez dayanma ilə, validasiya xətasının minimuma çatan kimi təlimi dayandırırsınız. Bu o qədər sadə və səmərəli nizamlama yöntəmidir ki, Geoffrey Hinton bunu "gözəl pulsuz nahar" adlandırırdı.



Şəkil 4-20. Tez dayanma nizamlanması

İPUCU

—-

Stoxastik və mini-toplu qradient eniş ilə, əyrilər o qədər də hamar deyil və minimuma çatıb-çatmadığınızı bilmək çətin ola bilər. Həll yolundan birisi, validasiya xətasının bir müddət minimumdan yuxarı olduqdan sonra dayandırmaqdır(modelin daha yaxşı olmayacağına əmin olduqdan sonra), sonra isə, model parametrlərini validasiya xətasının minimum olduğu nöqtəyə geti qaytarmaqdır.

—-

Tez dayanmanın sadə implementasiyası (tətbiqi):

```
from copy import deepcopy
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

X_train, y_train, X_valid, y_valid = [...]  # split the quadratic dataset

preprocessing = make_pipeline(PolynomialFeatures(degree=90, include_bias=False),
                              StandardScaler())
X_train_prep = preprocessing.fit_transform(X_train)
X_valid_prep = preprocessing.transform(X_valid)
sgd_reg = SGDRegressor(penalty=None, eta0=0.002, random_state=42)
n_epochs = 500
best_valid_rmse = float('inf')

for epoch in range(n_epochs):
    sgd_reg.partial_fit(X_train_prep, y_train)
    y_valid_predict = sgd_reg.predict(X_valid_prep)
    val_error = mean_squared_error(y_valid, y_valid_predict, squared=False)
    if val_error < best_valid_rmse:
```

```
        best_valid_rmse = val_error
        best_model = deepcopy(sgd_reg)
```

Bu kod, həm təlim çoxluğu üçün, həm də validasiya çoxluğu üçün, əvvəlcə polinom xüsusiyyətləri əlavə edir və bütün giriş xüsusiyyətləri ölçür (kod güman edi ki, siz, təlim çoxluğunu daha kiçik təlim çoxluqlara və validasiya çoxluqlara bölübsünüz). Sonra o, nizamlanmamış və kiçik öyrənmə dərəcəsinə sahib olan `SGDRegressor` modelini yaradır. Təlim dövrü zamanı, inkremental öyrənməni yerinə yetirmək üçün o, `fit()` funksiyası əvəzinə `partial_fit()` funksiyasını çağırır. Hər bir dövr də o, validasiya çoxluğu üzərində KOKX(Kök Orta Kvadrat Xətası) ölçür. Əgər o, gördüyü ən kiçik KOKX-dan daha kiçik bir KOKX ilə qarşılaşarsa, o modelin nüxsəsini (copy) `best_model` dəyişəninə mənimsədir. Bu təlimi dayandırmır, lakin təlimdən sonra ən yaxşı modelə qayıtmağa imkan yaradır. Qeyd edək ki o, modeli `copy.deepcopy()` funksiyası vasitəsi ilə kopyalayıb, çünki bu funksiya, həm modelin hiperparametrlərini, həm də təlimlənmiş parametrlərini kopyalayır. Bunun əksi olaraq, `sklearn.base.clone()` funksiyası sadəcə modelin hiperparametrlərini kopyalayır.

Logistik Reqressiya

Fəsil 1-də müzakirə etdiyimiz kimi, bəzi reqresiyya alqoritmləri təsnifat üçün istifadə edilə bilər (və əksinə). Logistik reqressiya (həmçinin "logit reqressiyası" da adlanır) adətən nümunənin müəyyən bir sinifə aid olma ehtimalını qiymətləndirmək üçün istifadə olunur (məsələn, elektron mekturub (email) spam olma ehtimalı nədir?). Əgər təxmin edilən ehtimal verilmiş həddən (adətən 50%) böyükdürsə, model nümunənin həmin sinifə aid olduğunu proqnozlaşdırır ("müsbət sinif" adlanır, "1" ilə işarələnir) və əks halda o, bunu aid olmadığını proqnozlaşdırır (yəni, "mənfi sinif"-ə aiddir, "0" ilə işarələnir). Bu onu ikili təsnifatçı edir.

Ehtimalların qiymətləndirilməsi

Beləliklə, logistik reqressiya necə işləyir? Xətti reqressiya modelində olduğu kimi, logistik reqressiya modeli də giriş xüsusiyyətlərin çəkili cəmini hesablayır (üstəgəl qərəz termi), lakin xətti reqressiya modeli kimi birbaşa nəticəni çıxartmaq əvəzinə, logistik reqressiya nəticənin logistikasını verir (Tənlik 4-13).

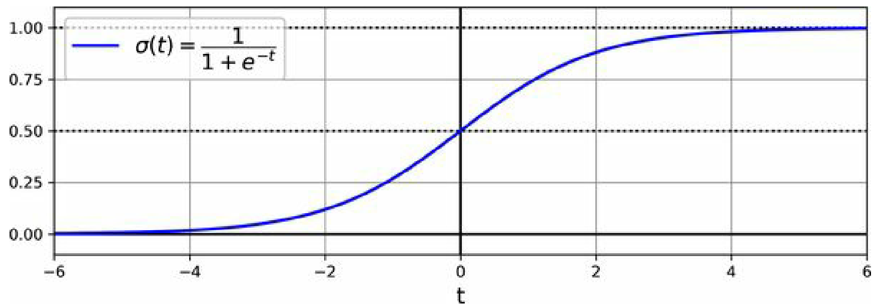
$$\hat{p} = h_{\theta}(\mathbf{x}) = \sigma\left(\boldsymbol{\theta}^{\top}\mathbf{x}\right)$$

Tənlik 4-13. Logistik reqressiya modelinin təxmin edilmiş ehtimalı (vektorlaşdırılmış formada)

Burada logistik - $\sigma(\cdot)$ kimi qeyd olunan - 0 ilə 1 arasında nəticə verən *sigmoid* funksiyasıdır (yəni, S-formalı). Tənlik 4-14. və Şəkil 4-21.-də olduğu kimi müəyyən edilir.

$$\sigma(t) = \frac{1}{1 + exp(-t)}$$

Tənlik 4-14. Logistik funksiya



Şəkil 4-21. Logistik funksiya

Logistik reqressiya modeli $\hat{p} = h_\theta(\mathbf{x})$ ehtimalını qiymətləndirdikdən sonra, hansı ki, $\mathbf{x}$ misalının müsbət sinifə aid olduğunu təyin edir, o, özünün $\hat{y}$ proqnozunu asanlıqla edə bilər (Tənlik 4-15).

$$\hat{y} = \begin{matrix} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{matrix}$$

Tənlik 4-15. 50% ehtimal həddindən istifadə edərək logistik reqressiya modelinin proqnozu

Diqqət yetirin ki, $t < 0$ olduqda $\sigma(t) < 0.5$ olur və $t \geq 0$ olduqda isə $\sigma(t) \geq 0.5$ olur, buna görə də, 50% ehtimal həddindən istifadə edən logistik reqressiya modeli $\boldsymbol{\theta}^\top \mathbf{x}$-i müsbət olduqda 1, neqativ olduqda isə 0 proqnozlaşdırır.

QEYD

—-

t hesabı tez-tez logit adlanır. Bu ad, logit funksiyasının $logit(p) = log(p \ / \ (1 - p))$ kimi təyin olunmasından və logistik funksiyasının tərsi olmasından irəli gəlir. Həqiqətəndə, p təxmini ehtimalının logit funksiyasını hesablasaz, nəticənin t olduğunu görəcəksiniz. Logit funksiyasına bəzən "*log-odds*" da deyilir, çünki o, müsbət sinifin texmin edilən ehtimalı ilə mənfi sinifin təxmin edilən ehtimal arasındakı nisbətin logudur.

TƏLİM VƏ DƏYƏR FUNKSİYASI

Siz logistik reqressiya modelinin ehtimalları necə qiymətləndirdiyini və necə proqnozlar verdiyini bilirsiniz. Bes bu necə təlimlənir? Təlimin məqsədi θ parametr vektorunu elə təyin etməkdir ki, model müsbət hallar üçün $(y = 1)$ yüksək ehtimalları və mənfi hallar üçün $(y = 0)$ aşağı ehtimalları qiymətləndirsin. Bu idea, dəyər funksiyası vasitəsi ilə $\mathbf{x}$ tək təlim dəyişə üçün Tənlik 4.16-da göstərilmişdir.

$$c(\theta) = \begin{matrix} -log(\hat{p}) & \text{if } y = 1 \\ -log(1 - \hat{p}) & \text{if } y = 0 \end{matrix}$$

Tənlik 4-16. tək təlim dəyişə üçün dəyər funksiyası

Bu dəyər funksiyası məna kəsb edir, çünki t sıfıra yaxınlaşdıqca $-log(t)$ çox böyüyür, buna görə də model, müsbət nümünələr üçün ehtimalı sıfıra yaxın təxmin edirsə, o zaman dəyəri böyük olacaq və əgər o, mənfi nümünələr üçün ehtimalı 1-ə yaxın təxmin edirsə, o zaman yenə də dəyər böyük olacaq. Digər tərəfdən, t 1-ə yaxınlaşdıqca $-log(t)$ sıfıra yaxınlaşır, buna görə də, ehtimalı mənfi nümunələr üçün təxmin edilən ehtimal sıfıra və ya müsbət nümunələr üçün təxmin edilən ehtimal birə yaxınlaşarsa, dəyəri sıfıra yaxın olacaq, bu da ki məhz bizim istədiyimiz nəticədir.

Dəyər funksiyası bütün təlim çoxluğu üzrə bütün təlim nümunələri üzrə orta qiymətdir. O, Tənlik 4-17-də göstərildiyi kimi "log itkisi" adlanan tək ifadə ilə yazıla bilər.

$$J(\boldsymbol{\Theta}) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{p}^{(i)}\right)$$

Tənlik 4-17. Logistik reqressiyasının dəyər funksiyası (log itkisi)

XƏBƏRDARLIQ

Log itkisi göydən düşməyib, onu riyazi olaraq "Bayesian inference"-i istifadə etməklə göstərmək olar. Bu da, itkini minimallaşdırır harada ki, nümunələr öz sinifinin ortası ətrafında "Gaussian" paylanmasına əməl etdiyini fərz etsək, modelin optimal olma ehtimalı ilə nəticələnəcək. Log itkisini istifadə etdiyiniz zaman bu, sizin etdiyiniz etirazsız fərziyyədir. Bu fərziyyə nə qədər yanlış olarsa, model bir o qədər qərəzli olar. Eynilə, xətti reqressiya modelini öyrətmək üçün OKX-dan (orta kvadrat xətası) istifadə etdikdə, məlumatların sırf xətti olduğunu və bəzi Gaussian səs-küyü olduğunu etirazsız şəkildə fərz edirik. Beləliklə, əgər data xətti dəyilsə (məsələn, kvadratdırsa) və ya səs-küy Gaussian dəyilsə (məsələn, kənar göstəricilər eksponensial nadir deyilsə), o zaman model qərəzli olacaqdır.

—

Pis xəbər odur ki, dəyər funksiyasını minimuma endirən θ dəyərini hesablamaq üçün bilinən hər hansı bir "closed-form" funksiyası mövcud deyil (normal tənliyin ekvivalenti yoxdur). Amma yaxşı xəbər odur ki, həmin dəyər funksiyası qabarıqdır, ona görə də, qradient eniş (və ya hər hansı bir başqa optimallaşdırma alqoritmi) qlobal minimumu tapmağa zəmanət verir (əgər öyrənmə dərəcəsi çox böyük dəyilsə və kifayət qədər uzun müddət gözləsəniz). j^{th} model parametri olan θ_j ilə bağlı olan dəyər funksiyasının qismən törəmələri Tənlik 4-18 ilə göstərilmişdir.

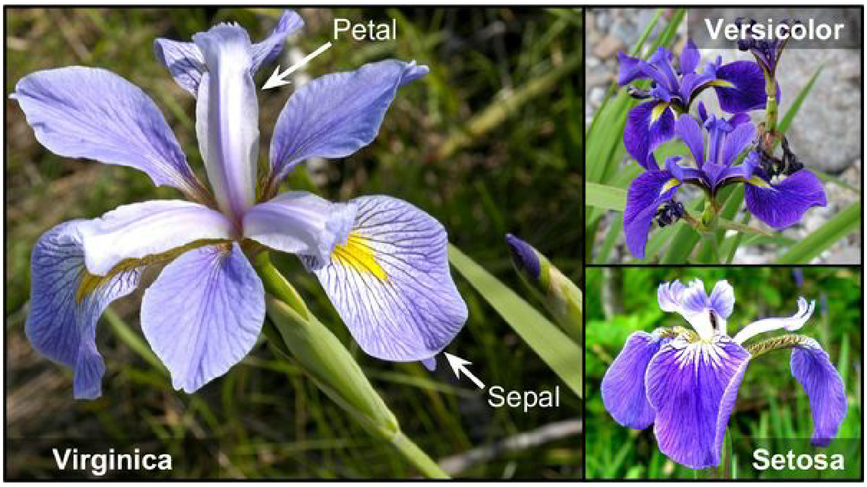
$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sigma\left(\boldsymbol{\theta}^\top \mathbf{x}^{(i)}\right)\left(-y^{(i)}\right) x_j^{(i)}$$

Tənlik 4-18. Logistik dəyər funksiyasının qismət törəmələri

Bu tənlik, Tənlik 4-5 çox bənzəyir: hər bir instans üçün o,proqnozlaşdırma xətasını hesablayır və onu j^{th} xüsusiyyətinin dəyərinə vurur, sonra isə bütün təlim instansları üzrə ortalamasını hesablayır. Bütün qismən törəmələri əhatə edən qradient vektorunuz varsa, o zaman siz onu TQE (toplu qradient eniş) alqoritmində istifadə edə bilərsiniz. Beləliklə, siz logistik reqressiya modelini necə təlim edəcəyinizi öyrəndiniz. Stoxastik QE üçün hər dəfə bir instans (nümunə) götürəcəksiniz, və mini-toplu QE üçün isə bir anda mini-toplusundan istifadə edəcəksiniz.

QƏRAR SƏRHƏDLƏRİ

Gəlin logistik reqressiyanı göstərmək üçün süsən (iris) çiçəyinin data çoxluğundan istifadə edək. Bu, ən məşhur olan məlumat toplusu olmaqla, *Iris setosa* , *Iris versicolor* və *Iris virginica* cinslərindən olan 150 süsən çiçəyinin sepal və ləçək uzunluğu və ənini özündə cəmləyir.



Şəkil 4-22. Üç süsən bitki növünün çiçəkləri

Gəlin yalnız ləçək eni xüsusiyyətinə əsaslanaraq "*Iris virginica*" növünü aşkar etmək üçün təsnifatı qurmağa çalışaq. İlk addım məlumatları yükləmək və göz atmaq:

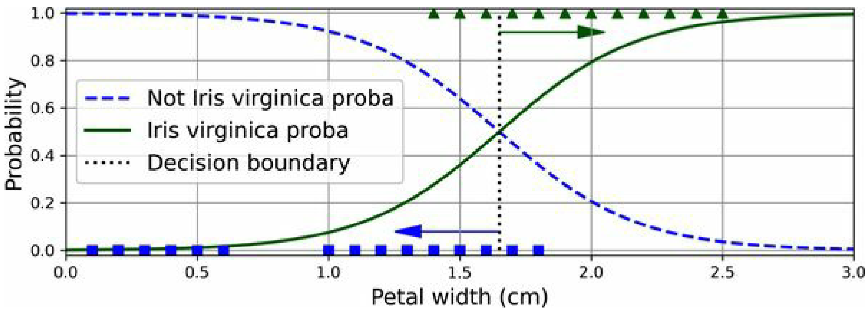
```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris(as_frame=True)
>>> list(iris)
['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
 'filename', 'data_module']
>>> iris.data.head(3)
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
>>> iris.target.head(3)  # note that the instances are not shuffled
0    0
1    0
2    0
Name: target, dtype: int64
>>> iris.target_names
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

Sonra, biz məlumatları böləcəyik və təlim çoxluqları vasitəsi ilə logistik reqressiya modelini təlimləndirəciyik:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X = iris.data[["petal width (cm)"]].values
y = iris.target_names[iris.target] == 'virginica'
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)
```

Ləçək eni 0 sm-dən 3sm-ə qədər dəyişən çiçəklər üçün modelin təxmin edilən ehtimallarına baxaq (Şəkil 4-23):

```
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)  # reshape to get a column vector
y_proba = log_reg.predict_proba(X_new)
decision_boundary = X_new[y_proba[:, 1] >= 0.5][0, 0]

plt.plot(X_new, y_proba[:, 0], "b--", linewidth=2,
         label="Not Iris virginica proba")
plt.plot(X_new, y_proba[:, 1], "g-", linewidth=2, label="Iris virginica proba")
plt.plot([decision_boundary, decision_boundary], [0, 1], "k:", linewidth=2,
         label="Decision boundary")
[...] # beautify the figure: add grid, labels, axis, legend, arrows, and samples
plt.show()
```

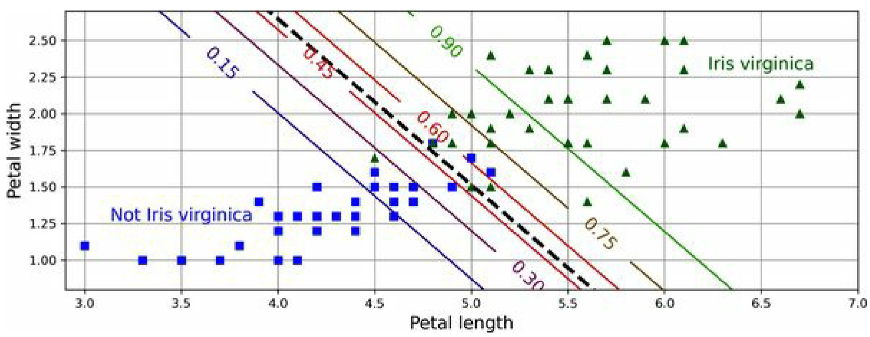


Şəkil 4-23. Təxmini ehtimallar və qərar sərhədi

İris virginica çiçəklərinin (üçbucaq şəklində təmsil olunur) ləçək eni 1.4 sm-dən 2.5 sm-ə qədərdir, lakin digər süsən (iris) çiçəklərinin (kvadratlarla təmsil olunur) isə ümumiyyətlə 0.1 sm-dən 1.8 sm arasında ləçək eninə sahibdirlər. Diqqət yetirsəz, onlar biraz üst-üstə düşürlər. Əgər 2 sm-dən yuxarıdırsa, klassifikator onun *İris virginica* çiçəyi olduğunu çox əmindir (həmin sinif üçün yüksək ehtimal verir), digər tərəfdən əgər 1 sm-dən aşağıdırsa, o zaman klassifikator onun *İris virginica* çiçəyi olmadığını əmindir ("Not Iris virginica" sinfi üçün yüksək ehtimal). Bu ikisinin arasında, klassifikator əmin cavab verə bilmir. Lakin, ondan sinifi proqnozlaşdırmasını istəsəniz (`predict_proba()` metoddunun əvəzinə `predict()` metodundan istifadə etsəz), o, daha çox ehtimal edilən sinifi qaytaracaq. Buna görə də, 1.6 sm ətrafında hər iki ehtimalın 50%-ə bərabər olduğu bir qərar sərhədi var: ləçək eni 1.6 sm-dən yuxarıdırsa, klassifikator onun *İris virginica* çiçəyi olduğunu təxmin edəcək, əks halda onun *İris virginica* çiçəyi olmadığını təxmin edəcək (çox əmin olmasa belə):

```
>>> decision_boundary
1.6516516516516517
>>> log_reg.predict([[1.7], [1.5]])
array([ True, False])
```

Şəkil 4-24. eyni data çoxluğunu, lakin bu dəfə fərqli iki xüsusiyyəti əks etdirir: ləçək eni və uzunluğunu. Təlimdən sonra, logistik reqressiya klassifikatoru bu iki xüsusiyyətə əsaslanaraq, yeni çiçəyin *İris virginica* çiçəyi olub olmadığını qiymətləndirə bilər. Modelin 50% ehtimal ilə qiymətləndirdiyi nöqtələri qırıq xəttlər təmsil edir: bu modelin qərar verdə sərhədidir. Diqqət yetirin ki, o, xətti sərhəddir. Hər bir parallel xətt nöqtələri təmsil edir, harada ki, model 15%-dən (aşağı sol) 90%-ə yuxarı sağ) qədər xüsusi ehtimalı çıxarır. Modelə görə, yuxarı sağ xəttdən kənarda olan bütün çiçəklərin *İris virginica* çiçəyi olma ehtimalı 90%-dən çoxdur.



Şəkil 4-24. Xətti qərarların sərhədləri

QEYD

Scikit-Learn LogisticReqresson modelinin nizamlanma gücünə nəzarət edən hiperparametrlər (digər xətti modellərdə olduğu kimi) alpha deyillər, əksinə tərsidirlər: C. C-nin dəyəri nə qədər yüksək olsa, model bir o qədər az nizamlanır.

—

Digər xətti modellər kimi, logistik reqressiya modelləri də ℓ_1 və ℓ_2 penaltilərdən istifadə etməklə nizamlana bilər. Scikit-Learn əslində standart olaraq ℓ_2 cərimələsini əlavə edir.

Softmax Reqressiyası

Logistik reqressiya modeli, çoxlu binar klassifikatorları öyrətmədən və birləşdirmədən, birbaşa çox sinifi dəstəkləmək üçün ümumiləşdirilə bilər (Fəsil 3-də müzakirə etdiymiz kimi). Buna softmax reqressiyası və ya multinominal logistik reqressiyası deyilir.

İdea çox sadədir: $\mathbf{x}$ dəyişəni verildikdə, softmax reqressiya modeli ilk olaraq hər bir k sinifi üçün, $s_k(\mathbf{x})$ dəyərini hesablayır, sonra isə softmax funksiyasını (həmçinin normallaşdırılmış eksponensial adlanır) hər bir dəyərə tətbiq etməklə sinifin ehtimalını proqnozlaşdırır. $s_k(\mathbf{x})$ tənliyi bizə tanışdır, çünki o, xətti reqressiyanın proqnozlaşdırılması üçün istifadə olunan tənliyə bənziyir (Tənlik 4-19).

$$s_k(\mathbf{x}) = \left(\theta^{(k)}\right)^{\top} \mathbf{x}$$

Tənlik 4-19. k sinifi üçün softmax dəyəri

Qeyd edək ki, hər bir sinifin özünəməxsus $\theta^{(k)}$ parametr vektoru var. Bütün vektorlar adətən $\mathbf{\Theta}$ parametr matrisində sətirlər şəklində saxlanılır.

$\mathbf{x}$ nümunəsinin hər bir sinifin xalını hesabladıqdan sonra, $\hat{p}_k$ ehtimalını təxmin edə bilərsiniz, hansı ki, xalları softmax funskiyası vasitəsilə icra etməklə, dəyişənin k sinifinə aid olma ehtimalını təxmin edə bilərsiniz (Tənlik 4-20). Funskiya hər xalın eksponensialını hesablayır, sonra isə onları normallaşdırır (bütün eksponensialların cəminə bölməklə). Nəticələr ümumiyyətlə logitlər və ya log-odds adlanır (baxmayaraq ki, onlar faktiki olaraq normallaşdırılmamış “log-odds”lardır).

$$\hat{p}_k = \sigma\mathbf{s}(\mathbf{x}))_k = \frac{\exp s_k(\mathbf{x}))}{\sum_{j=1}^{K} \exp s_j(\mathbf{x})\Big)}$$

Tənlik 4-20. Softmax funksiyası

Bu tənlikdə:

- K - siniflərin sayı
- $\mathbf{s}(\mathbf{x})$ - $\mathbf{x}$ dəyişənin hər bir sinifin dəyərini özündə saxlayan vektordur
- $\sigma\mathbf{s}(\mathbf{x}))_k$ - $\mathbf{x}$ dəyişənin k sinifinə aid olmasının təxmini ehtimalıdır, həmin nümunənin hər bir sinifi üçün verilən dəyərləri nəzərə almaqla.

Logistik reqressiya klassifikatoru kimi, Tənlik 4-21-də göstərildiyi kimi, standart olaraq softmax reqressiya klassifikatoru, ən yüksək təxmin edilən ehtimala malik sinifi proqnozlaşdırır (bu sadəcə olaraq, ən yüksək dəyər sahib olan sinifidir).

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}} \left(\left(\boldsymbol{\theta}^{(k)}\right)^{\top} \mathbf{x}\right)$$

Tənlik 4-21. Softmax reqressiya klassifikatorun proqnozu

argmax operatoru funksiyanı maksimallaşdıran dəyişənin dəyərini qaytarır. Bu tənlikdə o, *k* dəyərini qaytarır, hansı ki, $\sigma\mathbf{s}(\mathbf{x}))_k$ tehmini ehtimalını maksimallaşdırır.

İPUCU

Softmax reqressiya klassifikatoru eyni vaxtda yalnız bir sinifi proqnozlaşdırır (yəni bu, çoxsiniflidir, çox çıxışlı deyil), ona görə də o, yalnız bir-birini istisna edən siniflərlə istifadə edilməlidir, misal üçün müxtəlif növ bitkilər kimi. Siz bunu, bir şəkildə birdən çox insanı tanımaq üçün istifadə edə bilməzsiniz.

—-

İndi siz modelin necə qiymətləndirdiyini və proqnozların necə verildiyini bildiyiniz üçün, gəlin necə təlim olunduğuna nəzər yetirək. Məqsəd, hədəf sinif üçün yüksək ehtimalı (amma, digər siniflər üçün aşağı ehitmalı) təqdim edən modelə sahib olmaqdır. Tənlik 4-22-də göstərilən dəyər funksiyasının minumuma endirilməsi "çarpaz entropiya" adlanır. O hədəf sinfi üçün aşağı ehtimalı təxmin etdikdə modeli cəzalandırdığı üçün bu məqsədə gətirib çıxarmalıdır. Çarpaz entropiya tez-tez təxmin edilən sinif ehtimallarının hədəf siniflərə nə qədər uyğun olduğunu ölçmək üçün istifadə olunur.

$$J(\mathbf{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

Tənlik 4-22. Çarpaz entropiya dəyər funksiyası

Bu tənlikdə, i^{th} dəyişənin k sinifinə aid olma ehtimalı $y_k^{(i)}$ ilə ifadə olunub. Ümumiyyətlə, nümunənin sinifə aid olub-olmamasından asılı olaraq ya 1, və ya 0 dəyərinə bərabərdir.

Diqqət yetirin ki, yalnız iki sinif olduqda ($K = 2$), bu dəyər funksiyası logistik reqressiya dəyər funksiyasına bərabərdir (log itkisi; Tənlik 4-17-yə bax).

Çarpaz Entropiya

Çarpaz entropiya, Claude Shannon-un məlumat nəzəriyyəsindən yaranmışdır. Təsəvvür edin ki, siz hər gün hava haqqında məlumatı səmərəli şəkildə ötürmək istəyirsiniz. Əgər mümkün səkkiz variant olsa (günəşli, yağışlı və s.), o zaman siz hər bir variantı 3 bitdən istifadə etməklə kodlaya bilərsiniz, çünki $2^3 = 8$. Lakin, düşünürsünüzsə hər gün günəşli olacaq, o zaman səmərəli olması üçün "günəşli" variantı bir bitlə (0), digər yeddi variantı isə 4 bit ilə kodlaşdırmaq olar (1-dən başlayaraq). Çarpaz entropiya hər seçimə göndərdiyiniz bitlərin orta sayısını ölçür. Əgər hava haqqında təxminləriniz mükəmməldirsə (tam doğrudursa), o zaman çarpaz entropiya havanın özünün entropiyasına bərabər olacaq (yəni, onun daxili gözlənilməzliyi). Lakin, fərziyyəniz səhvdirsə (məsələn, tez-tez yağış yağarsa), çarpaz entropiya, *Kullback-Leibler (KL) divergensiyası* adlanan miqdardan böyük olacaqdır.

İki p və q ehtimal paylanması arasında çarpaz entropiya, $H(p,q) = -\Sigma_x p(x) \log q(x)$ ifadəsi ilə müəyyən edilir (ən azı paylanmalar diskret olduqda). Daha etraflı, video materiala baxa bilərsiniz.

—-

$\theta^{(k)}$ ilə bağlı olan dəyər funksiyasının qradient vektoru, Tənlik 4-23 ilə verilmişdir.

$$\nabla_{\theta^{(k)}} J(\mathbf{\Theta}) = \frac{1}{m}\sum_{i=1}^{m} \hat{p}_k^{(i)} - y_k^{(i)} \Bigg) \mathbf{x}^{(i)}$$

Tənlik 4-23. k sinifi üçün çarpaz entropiya qradient vekotur

İndi siz hər bir sinif üçün qradient vektoru hesablaya bilərsiniz, sonra isə dəyər funksiyasını minimuma endirən **Θ** parametr matrisini tapmaq üçün qradient enişdən (və ya hır hansı digər optimallaşdırma alqoritmindən) istifadə edə bilərsiniz.

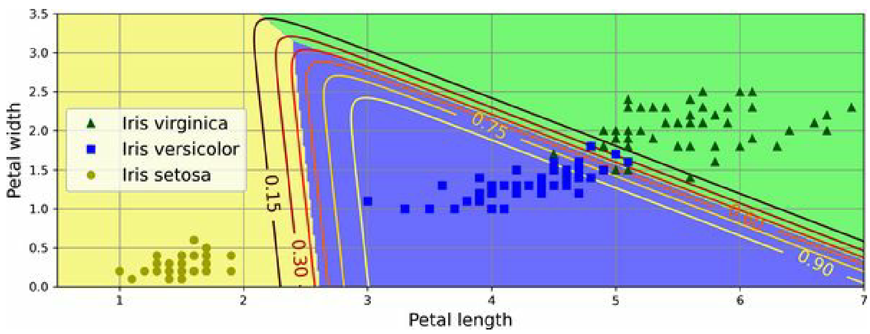
İris bitkilərini hər üç sinif vasitəsilə təsnif etmək üçün softmax reqressiasından istifadə edək. Scikit-Learn-in `LogisticRegression` klassifikatoru ikidən çox sinif ilə təlim etdiyiniz zaman softmax reqressiyasından avtomatik istifadə edir (fərz etsək ki solver="lbfgs" istifadə olunur, bu isə default). O, həmçinin, əvvəllər qeyd edildiyi kimi, defolt olaraq ℓ_2 nizamlamasını tətbiq edir, hansını ki, C hiperparametrdən istifadə edərək idarə edə bilərsiniz:

```
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = iris["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

softmax_reg = LogisticRegression(C=30, random_state=42)
softmax_reg.fit(X_train, y_train)
```

Beləliklə, növbəti dəfə ləçəklərin uzunluğu 5 sm və eni 2 sm olan iris tapdığınız zaman modelinizdən onun hansı irisin növü olduğunu söyləməsini xahiş edə bilərsiniz və o, 96% ehtimalla *Iris virginica* (2-ci sinif) cavab verəcək (və ya *Iris versicolor* 4% ehtimalla):

```
>>> softmax_reg.predict([[5, 2]])
array([2])
>>> softmax_reg.predict_proba([[5, 2]]).round(2)
array([[0.  , 0.04, 0.96]])
```

Şəkil 4-25 fon rengləri ilə təmsil olunan nəticədə qərar sərhədlərini göstərir. Diqqət yetirin ki, istənilən iki sinif arasındakı qərar sərhədləri xəttidir. Şəkildə həmçinin əyri xəttlərlə təsvir olunan xəttlər *Iris versicolor* sinifi üçün ehtimalları göstərir (məsələn, 0.30 ilə işarələnmiş xətt, 30% ehtimal sərhədini təmsil edir). Diqqət yetirin ki, model təxmini ehtimalı 50%-dən aşağı olan bir sinifi proqnozlaşdıra bilər. Məsələn, bütün qərar sərhədlərinin qovuşduğu nöqtədə, bütün siniflər 33% bərabər ehtimala malikdirlər.



Şəkil 4-25. Softmax reqressiyanın qərar sərhədləri

Bu fəsildə siz həm reqressiya, həm də klassifikasiya üçün xətti modelləri təlimləndirməyin müxtəlif yollarını öyrəndiniz. Siz xətti reqressiyasnı, eləcə də qradient enişi həll etmək üçün qapalı formalı tənlikdən istifadə etdiniz və modeli nizama salmaq üçün təlim zamanı dəyər funksiyasına müxtəlif cərimələrin necə əlavə oluna biləcəyini öyrəndiniz. Siz həmçinin öyrənmə əyrilərini necə tərtib etməyi və onları təhlil etməyi və erkən dayandırmağı necə həyata keçirməyi öyrəndiniz. Nəhayət, siz logistik reqressiya və softmax reqressiyasının necə işlədiyini öyrəndiniz. Biz ilk maşın öyrənmənin qara qutularını açdıq! Növbəti fəsillərdə dəstək vektor maşınlarından başlayaraq daha çoxunu açacağıq.

Misallar

1. Milyonlarla funksiyaya malik təlim çoxluğunuz varsa, hansı xətti reqressiya təlim alqoritmindən istifadə edə bilərsiniz?

2. Tutaq ki, təlim çoxluğunda funksiyalar çox fərqli yüklərə malikdir. Hansı alqoritmlər bundan əziyyət çəkə bilər və necə? Bununla bağlı nə edə bilərsiniz?
3. Logistik reqressiya modelini öyrədərkən gradient enişi daxili minimumda ilişib qala bilərmi?
4. Bütün gradient eniş alqoritmləri kifayət qədər uzun müddət işləməsinə icazə vermək şərti ilə eyni modelə gətirib çıxarırmı?
5. Tutaq ki, siz toplu gradient enişindən istifadə edirsiniz və hər dövrdə validasiya xətasını tərtib edirsiniz. Validasiya xətasının davamlı olaraq artdığını görsəniz, nə baş verər? Bunu necə düzəldə bilərsiniz?
6. Validasiya xətası artdıqda, mini-toplu gradient enişini dərhal dayandırmaq yaxşı fikirdirmi?
7. Hansı gradient eniş alqoritmi (müzakirə etdiyimiz alqoritmlər arasında) optimal həllin yaxınlığına daha tez çatacaq? Hansı əslində birləşəcək? Başqalarını da necə birləşdirə bilərsiniz?
8. Tutaq ki, polinom reqressiyasından istifadə edirsiniz. Siz öyrənmə əyrilərini tərtib edirsiniz və tılim xətası ilə validasiya xətası arasında böyük bir boşluq olduğunu görürsünüz. Nə baş verir? Bunu həll etməyin üç yolu nədir?
9. Tutaq ki, siz silsilənin reqressiyasından istifadə edirsiniz və siz təlim xətasının və validasiya xətasının demək olar ki, bərabər və kifayət qədər yüksək olduğunu görürsünüz. Modelin yüksək qərəzli və ya yüksək dispersiyadan əziyyət çəkdiyini söyləyərdiniz? α regulyasiya hiperparametrini artırmaq və ya azaltmaq lazımdır mı?
10. Niyə istifadə etmək istərdiniz:
    a. Düz xətti reqressiya əvəzinə silsilə reqressiyası (yəni heç bir nizamlanma olmadan)
    b. Silsilə reqressiyasının əvəzinə lasso reqressiyası?
    c. lasso reqressiyasının əvəzinə elastik şəbəkə reqressiyası?
11. Tutaq ki, siz şəkilləri açıq hava/qapalı və gündüz/gecə kimi təsnif etmək istəyirsiniz. İki logistik reqressiya təsnifatını və ya bir softmax reqressiya təsnifatını tətbiq edərdiniz?
12. Scikit-Learn istifadə etmədən (yalnız NumPy) softmax reqressiyası üçün erkən dayandırmaqla toplu gradient enişini həyata keçirin. Onu iris verilənlər toplusu kimi təsnifat tapşırığında istifadə edin.

Bu misalların həlli bu fəslin sonunda, https://homl.info/colab3 ünvanında mövcuddur .

---

1. Qapalı forma tənliyi yanlız sonlu sayda sabitlər, dəyişənlər və standart əməliyyatlardan ibarətdir: məsələn, $a = sin(b - c)$. Sonsuz cəmlərsiz, məhdudiyyətsiz, inteqrallarsız və s.
2. Texniki baxımdan, onuntörəməsi “Lipschitz continuous”-dır.
3. 1ci xüsusiyyət daha kiçik olduğundan, dəyər funksiyasına təsir etmək üçün θ_1-də böyük dəyişiklik tələb olunur, buna görə də qab θ_1 oxu boyunca uzanır.
4. Eta (η) yunan əlifbasının yedinci hərfidir.
5. Normal tənlik yalnız xətti reqressiyanı yetirə bildiyi halda, qradient eniş alqoritmləri bir çox digər modelləri öyrətmək üçün istifadə edilə bilər.
6. Bu qərəz anlayışı xətti modellərin qərəzli termini ilə qarışdırılmamalıdır.
7. Qısa adı olmayan dəyər funksiyaları üçün $J(\theta)$ qeydindən istifadə etmək adi haldır; Kitabın qalan hissəsində tez-tez bu qeyddən istifadə ediləcək. Məzmun hansı dəyər funskiyasından müzakirə edildiyini aydınlaşdıracaq.
8. Normalar 2ci fəsildə müzakirə olunub.
9. Əsas diaqonalda 1-lər istisna olmaqla, 0-larla dolu kvadrat matris (yuxarı soldan sağa).

10. Alternativ olaraq, Ridge sinifindən "sag" helledici ilə birgə istifadə edə bilərsiniz. Stoxastik ortalama qradient eniş, qradient enişin bir növüdür. Daha ətraflı, British Columbia universitetindən Mark Schmidtin "Minimizing Finite Sums with the Stochastic Average Gradient Algorithm" təqdimatına baxa bilərsiniz.
11. Differensiallaşmayan nöqtədəki subqradient vektorunu, həmin nöqtənin ətrafındakı qradient vektorları arasında ara vektor kimi düşünə bilərsiniz.
12. Fotolar müvafiq Vikipediya səhifələrindən götürülüb. Frank Mayfield (Creative Commons BY-SA 2.0) tərəfindən *Iris virginica* fotoşəkili , D. Gordon E. Robertson tərəfindən *iris rəngli fotoşəkil (* Creative Commons BY-SA 3.0), *Iris setosa* fotoşəkili ictimai sahə.
13. NumPy kitabxanasının `reshape()` funksiyası bir ölçünün –1 olmasına imkan verir ki, bu da "avtomatik" deməkdir: dəyər massivin uzunluğundan və qalan ölçülərdən çıxarılır.
14. Bu, $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$ olan $\mathbf{x}$ nöqtələrinin çoxluğudur, hansı ki, düz xətti təyin edir.