Аналитика больших массивов данных

(Алгоритмы машинного обучения)

семинар 2

1)Модель алгоритмов

$A = \{a(x, \alpha) \mid \alpha \in \}$, где $a: \times \to$ – фиксированная функция

2)Функция потерь (loss function):

$$(y, z)$$

3) Эмпирический риск

$$(a, X^{\ell}) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i), y_i)$$

4)Метод обучения:

$$\mu: \times \to$$

$$(X^{\ell}) = \arg\min_{a \in A} (a, X^{\ell})$$

Линейная регрессия

Пусть уже есть 10 объектов, ответы на каждом из объектов ,

и 3 характеристики. Хотим восстановить .

– матрица объект-признак (10 x 3)

– вектор столбец ответов (10 x 1)

Модель алгоритмов. Если – объект, то

$$a(x) = w_0 + w_1 \cdot x^1 + w_2 \cdot x^2 + w_3 \cdot x^3$$

где верхний индекс это номер характеристики.

Функция потерь

$$Q = (a(x) - y)^2$$

Обозначим за [illegible] – вектор столбец 10 на 1, у которого на всех позициях 1.

[illegible] = [illegible]

То есть припишем к нашей матрице объект признак слева столбец с 1.

[illegible] = [illegible]

Где [illegible] = [illegible], [illegible] -ответы алгоритма

Эмпирический риск (с точностью до индекса):

[illegible] – [illegible] ([illegible] – [illegible])

Имеем:

[illegible] – [illegible] [illegible] – [illegible]

Неизвестные только θ, после нахождения которых полностью восстановится алгоритм.

Аналитическое решение:

$$\theta = (X^T X)^{-1} X^T y$$

X:

```
[[ 1.    0.146 -1.157  0.379]
 [ 1.    1.286  0.658 -1.769]
 [ 1.   -1.872 -0.426 -0.022]
 [ 1.    0.949  0.958  0.694]
 [ 1.   -1.985  0.727  1.181]
 [ 1.    0.741 -0.393  0.083]
 [ 1.   -1.194 -1.97  -1.347]
 [ 1.    1.474  0.899 -1.009]
 [ 1.    0.193 -0.171  0.072]
 [ 1.   -0.643 -2.255  1.427]]
```

Y:

```
[ 0.87098153
 -3.70525948
 -1.57607053
  3.96241584
  2.9817153
  0.86255406
 -6.18732382
 -0.97511125
  0.50508739
  2.42358325]
```

```
X_full_tr = np.transpose(X_full)
teta_analitics = np.dot(np.dot(np.linalg.inv( np.dot(X_full_tr, X_full) ), X_full_tr), Y)
print teta_analitics
```

```
[ 0.31749375  0.78032738  0.74979223  3.08462768]
```

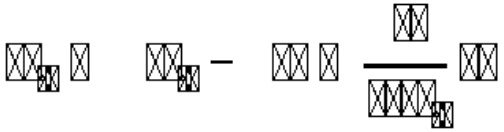
Аналитический подход:

-большая сложность

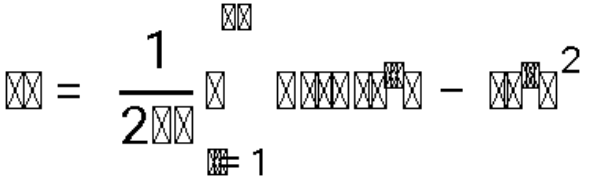
-не всегда возможен

Градиентный спуск

Итеративный метод.



Градиентный спуск для линейной регрессии:



– количество элементов в выборке

– i-ый объект

– i-ый ответ

– j-ая характеристика i-ого объекта

Тогда

$$\frac{\partial}{\partial \mathbf{w}_k} = \frac{\partial}{\partial \mathbf{w}_k} \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \mathbf{w}^T \mathbf{x}_i \right)^2 = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \mathbf{w}^T \mathbf{x}_i \right) \left(-x_{ik} \right)$$

Повторяем пока не сойдется:

$$w_k = w_k - \eta \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \mathbf{w}^T \mathbf{x}_i \right) \left(-x_{ik} \right)$$

В матричном виде:

$$\mathbf{w} = \mathbf{w} - \eta \frac{1}{n} \left(\mathbf{X}^T \left(\mathbf{X} \mathbf{w} - \mathbf{y} \right) \right)$$

```
teta_grad_start = np.zeros(4)
learning_rate = 0.001
epsilon = 0.00001
max_iter = 100000
def step(teta):
    return teta - learning_rate * (1.0/10) * np.dot(np.transpose(X_full), (np.dot(X_full, teta) - Y))
teta_GD = teta_grad_start
for i in range(max_iter):
    teta_GD_new = step(teta_GD)
    if np.sum(np.abs(teta_GD_new - teta_GD)) <= epsilon:
        break
    else:
        teta_GD = teta_GD_new
print i
print teta_GD
```

```
8454
[ 0.31673068  0.77492892  0.75208069  3.07801993]
```

```
X = np.around(np.random.normal(0, 1, [10, 3]), 3)
teta_real = np.random.normal(0, 1, 4)
print teta_real
```

```
[ 0.41517216  0.74724113  0.76996725  3.03290039]
```

```
Y = np.dot(X_full, teta_real) + np.random.normal(0, 0.1, 10)
```

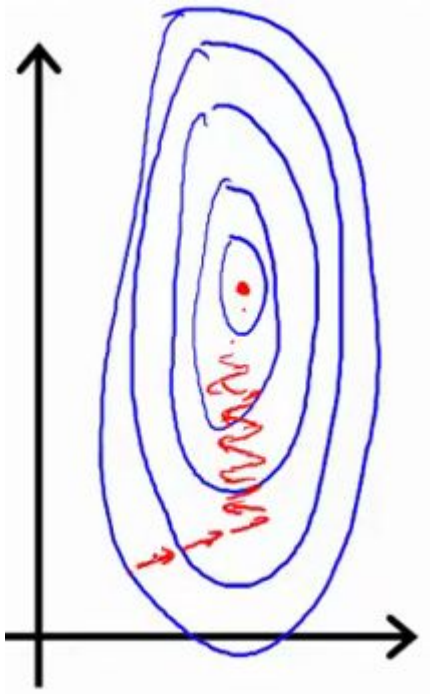
```
error_analitic = (np.abs(teta_analitics - teta_real))
print error_analitic
```

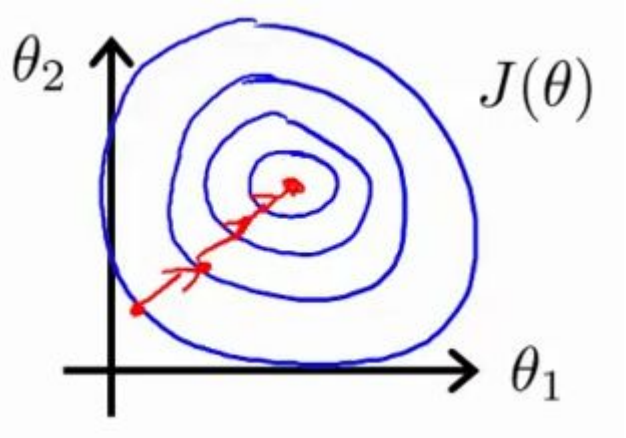
```
[ 0.09767841  0.03308625  0.02017502  0.05172729]
```

```
error_GD = (np.abs(teta_GD - teta_real))
print error_GD
```

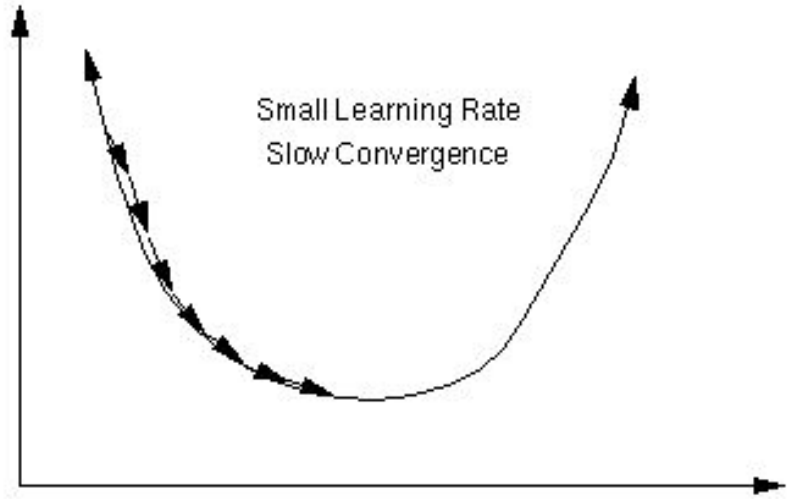
```
[ 0.09844148  0.02768779  0.01788656  0.04511954]
```

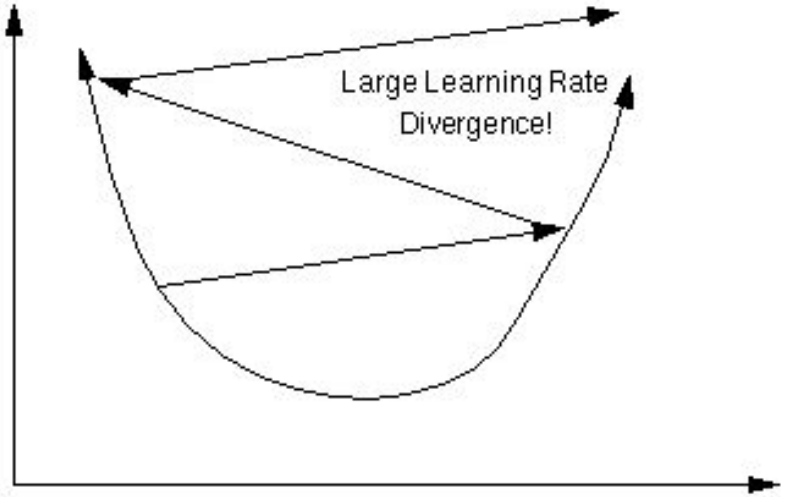
Градиентный спуск
Нормализация признаков



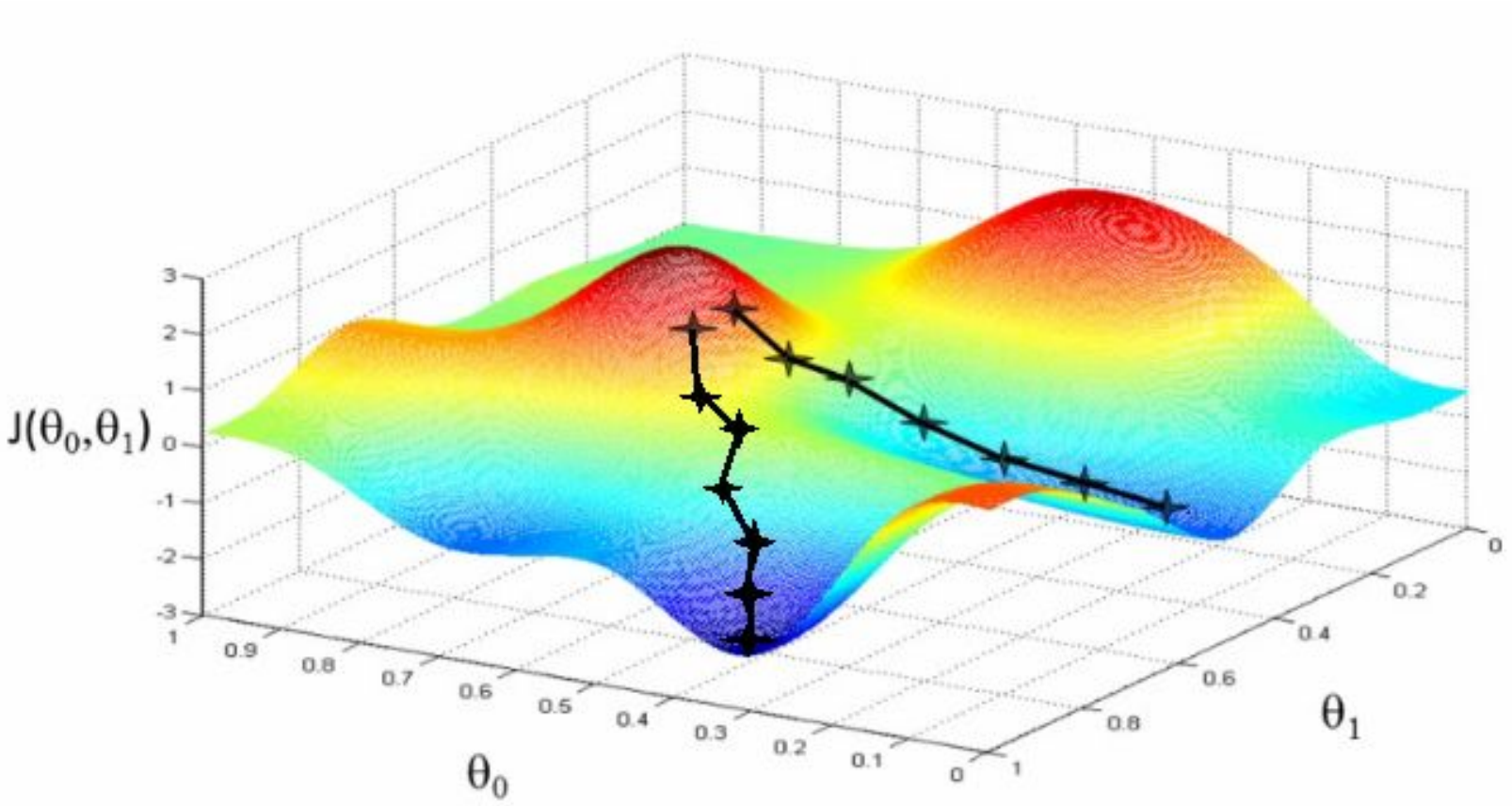


Выбор α


Small Learning Rate
Slow Convergence


Large Learning Rate
Divergence!

Локальный минимум



Возможное решение: запускать несколько раз стартуя с разных значений

Итого градиентный спуск.

-подбор многих параметров
    -learning rate
    -максимальное число итераций
    -условие сходимости
    -стартовые параметры (сетка)
-легко реализуем
-хорошо работает на больших выборках
-нет уверенности в том что выдал minimum функции

Подходит для большинства моделей.
Используется (с некоторыми усовершенствованиями) в том числе нейронных сетях.

Метрические методы классификации

Задача классификации.

$X^l = (X_i, Y_i)^l{}_{i=1}$ – обучающая выборка.

Гипотеза компактности: *Схожие объекты лежат в одном классе.*

Функция расстояния:

ρ: $X^2 \to [0, +\infty)$ (тождество, симметрия, неравенство треугольника). Например Евклидово расстояние

$$\rho(u, x_i) = \sum_{j=1}^{n} \left(\left| u^j - x_i^j \right|^2 \right)^{1/2}$$

Для элемента u сортируем объекты по расстоянию до u (от самого близкого, до самого дальнего). $x_u^{(i)}$ - i-ый сосед объекта среди выборки.

Метрический алгоритм классификации

$$\boldsymbol{a(u; X^l) = argmax_{\{y \epsilon Y\}} \sum_{i=1}^{l} \left[y_u^{(i)} = y \right] * w(i, u)}$$

Где $w(i, u)$ – вес (степень важности) i-ого соседа объекта u. Неотрицательное и не возрастает.

$\Gamma_i(u) = \sum_{i=1}^{l} \left[y_u^{(i)} = y \right] * w(i, u)$ оценка близости объекта к классу.

Метод ближайшего соседа

$$w(i,u) = [i = 1]$$

-прост в реализации
-интерпретируемость решения

-неустойчивость к погрешностям
-нет настраиваемых параметров
-низкое качество классификации
-необходимо хранить всю выборку целиком

Метод k ближайших соседей

$$w(i,u) = [i \leq k]$$

-менее чувствителен к шуму
-появился параметр k

Проблема: неоднозначность классификации при $\Gamma_y(u) = \Gamma_s(u), u \neq s$.