Aprendizaje Automático sobre Grandes Volúmenes de Datos

Clase 4

Pablo Ariel Duboue, PhD

Universidad Nacional de Córdoba,
Facultad de Matemática, Astronomía y Física

Material de lectura

- Clase pasada:
  - Capítulos 3 y 6 del Mitchel (1997)
- Ésta clase:
  - Gale, William A. (1995). "Good–Turing smoothing without tears". Journal of Quantitative Linguistics 2: 3. doi:10.1080/09296179508590051
  - Capítulo 5 del Marlsand (2009) "Machine Learning, an Algorithmic Perspective"
  - Capítulo 5 del Smola & Vishwanathan (2008) "Introduction to Machine Learning"
  - http://en.wikipedia.org/wiki/Logistic_regression
  - http://en.wikipedia.org/wiki/Linear_regression

Preguntas

- Cantidad de datos negativos?
  - Estimación de priors, muchas veces por fuera del aprendizaje automático en sí (análisis de datos)
- Pipeline de trabajo para clasificación: próxima clase
- Random forests: vamos a tener una clase dedicada a ellos en la segunda parte de la materia

Comentarios del feedback

- Palabras finales sobre word2vec: ejemplo de features interesantes, pero no es parte de la clase
- Aplicaciones específicas: fuera del alcance de la materia
- Feedback es obligatorio y firmado, incluyan si son alumnos de grado, posgrado u oyentes
- Material más introductorio: algunos alumnos están cursando MOOCs de aprendizaje automático, si pueden compartir sus experiencias en la lista de correo

NB: de conteos a probabilidades

- Conteos de instancias (N_i) vs. conteos de features (N_f)
- De la definición de probabilidad conjunta:
  $P(".com"|Arts)P(Arts) = P(".com", Arts) = \frac{N_f(".com", Arts)}{N_f(.)}$
- De la definición de probabilidad simple:
  $P(Arts) = \frac{N_i(Arts)}{N_i(.)}$
- Despejando para la probabilidad condicional:
  $P(".com"|Arts) = \frac{N_i(.)}{N_f(.)} \frac{N_f(".com", Arts)}{N_i(Arts)}$

Ejemplo de NB

1

```
$git clone https://github.com/DrDub/urlclassy.git
$ node
> eval(fs.readFileSync("example/features.js").toString());
eval(fs.readFileSync("example/classifier.js").toString())
> trained_classifier.totalExamples
373260
> trained_classifier.classTotals.Arts
18732
> trained_classifier.classTotals.Business
18477
```

NB: Conteos

Arte:

www.	13957
ww.a	1967
w.au	56
.aut	11
auto	6
utop	3
topa	2
opar	9
part	42
arts	150
rts.	70
ts.c	128
s.co	2567
.com	13203

Negocios:

www.	17115
ww.a	1517
w.au	64
.aut	24
auto	49
utop	2
topa	2
opar	5
part	73
arts	48
rts.	84
ts.c	392
s.co	3581
.com	14495

Ejemplo de NB

2

```
> var p_arts = trained_classifier.classTotals.Arts /
trained_classifier.totalExamples; p_arts
0.050184857739913 2
> var p_business = trained_classifier.classTotals.Business /
trained_classifier.totalExamples; p_business
0.049501687831538 34
> var ml_arts=1.0; for(var i=0; i<w.length;i++){var
y=trained_classifier.classFeatures.Arts[trained_features[w[i]]];
if(y){ml_arts*=y / trained_classifier.classTotals.Arts}}; ml_arts
1.6008344829669292e-32
> var ml_business=1.0; for(var i=0; i<w.length;i++){var
y=trained_classifier.classFeatures.Business[trained_features[w[i]]];
if(y){ml_business*=y/ trained_classifier.classTotals.Business}};
ml_business
4.330609317975143e-31
```

Estimando datos ausentes: smoothing

- Estimar probabilidades a partir de conteos tiene el problema de que muchos datos no son observados
  - ¿Qué hacer si un feature nunca aparece con un valor particular de la clase objetivo?
  - Técnicas de smoothing: quitar masa de probabilidad de los eventos observados para dársela a los eventos no observados
  - Sin smoothing la multiplicación de Naive Bayes da cero en muchos casos
- Opciones sencillas:
  - Lagrangiano: todo evento no observado se considera ocurre una vez
  - ELE:agregar 0.5 a todos los conteos
  - Add-tiny: agregar un número muy pequeño a todos los conteos

Simple Good-Turing

Frecuencia	Frecuencia de la frecuencia
r	N_r
1	120
2	40
3	24
4	13
5	15
6	5
7	11
8	2
9	2
10	1
11	0
12	3

Estimador r^*

- Si $N = \sum rN_r$, queremos estimar la probabilidad p_r para los objetos que vimos r veces:

$$p_r \equiv \frac{r^*}{N}$$

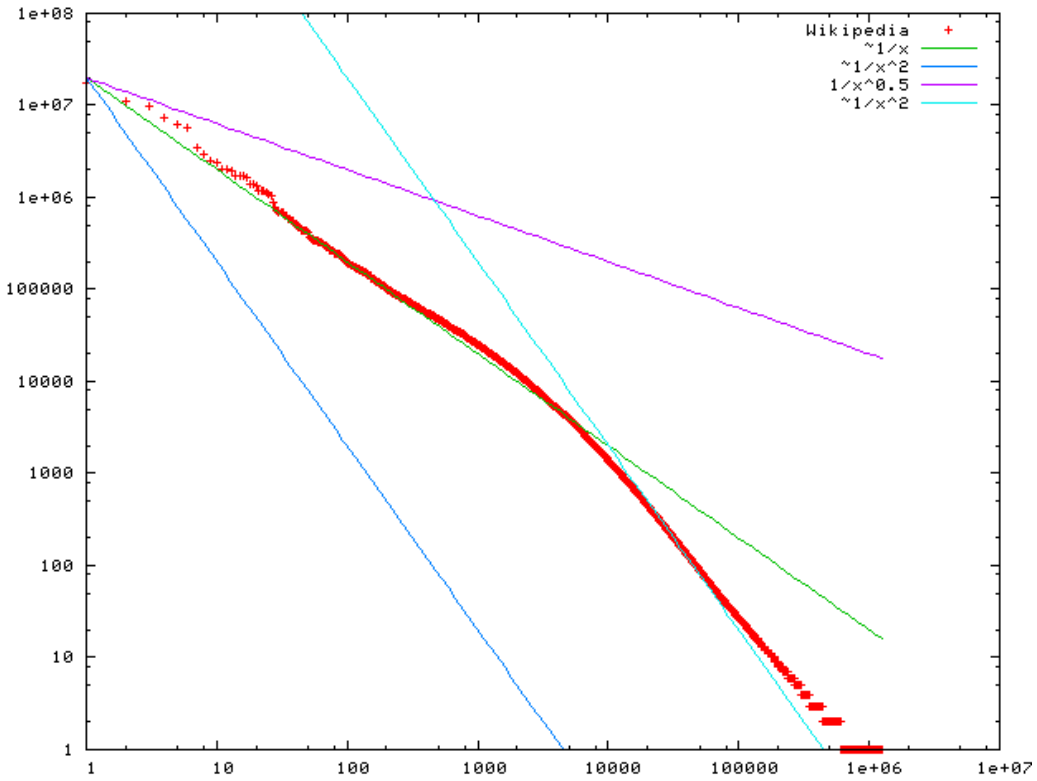
- Un estimador MLE será $p_r = r/N$ (o sea $r^* = r$) y predice la probabilidad de los elementos ausentes como cero (no es muy útil)
- El estimador Good-Turing utiliza:

$$r^* = (r+1)\frac{E(N_{r+1})}{E(N_r)}$$

- Good-Turing estima la probabilidad ausente como N_1/N
  - N_1es la frecuencia de frecuencias mejor medida por lo que $E(N_1) = N_1$es una buena aproximación

Distribución de Zipf

Frecuencia de palabras en Wikipedia:

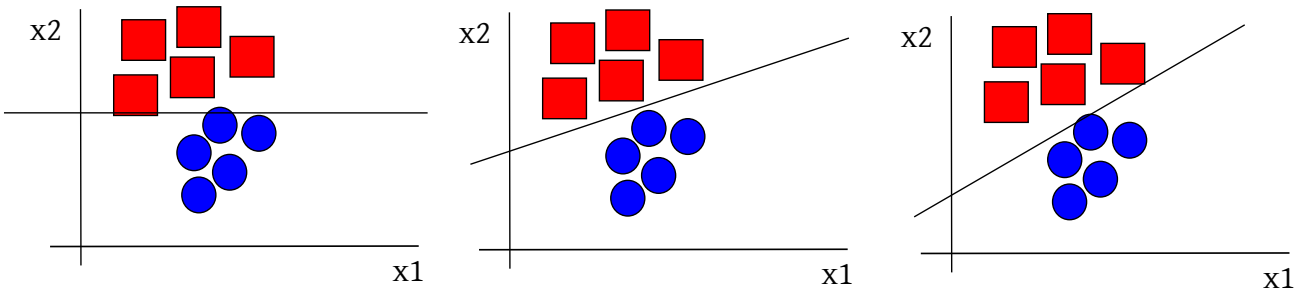


(LGPL por Victor Grishchenko)

Usándolo en la práctica

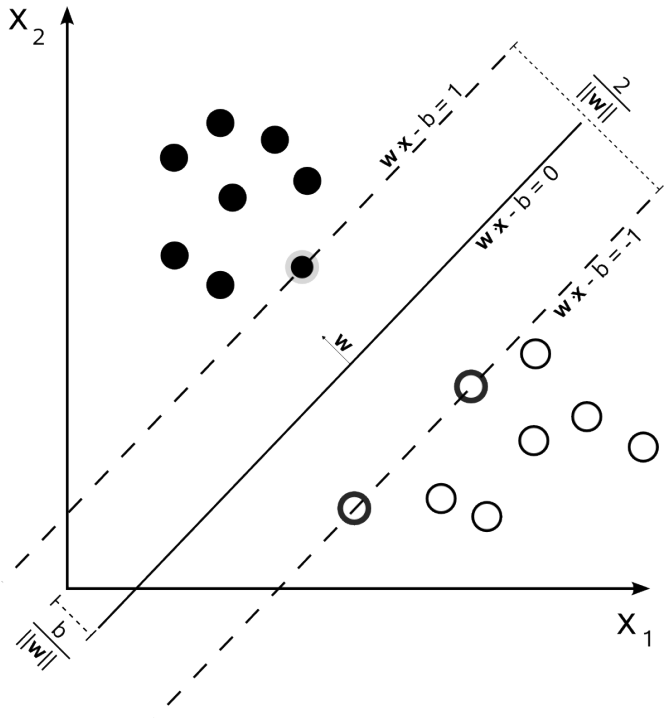
- Ajustar la probabilidad de los eventos observados usando los nuevos estimadores
- Distribuir la masa de probabilidad para los eventos no observados (N_1/N) a medida que aparecen
  - ¿Pero cuántos hay?
  - ¿Cómo distribuir esta probabilidad entre ellos?
- Si es posible, se usa conocimiento extra sobre la estructura de los eventos
  - Por ejemplo, si los eventos son pares de palabras, podemos estimar qué tan importante es un par que nunca vimos en función de cuán frecuentes son las palabras que lo componen

Separador de gran márgen



Un separador de gran márgen: mayor expectativa de mejor generalización

Intución



- Optimizar $\mathbf{w} \cdot \mathbf{x} - b = 0$ se puede resolver vía optimización de programación cuadrática
  - Si no existe hiperplano de separación, proyectamos a más dimensiones donde es más fácil que exista

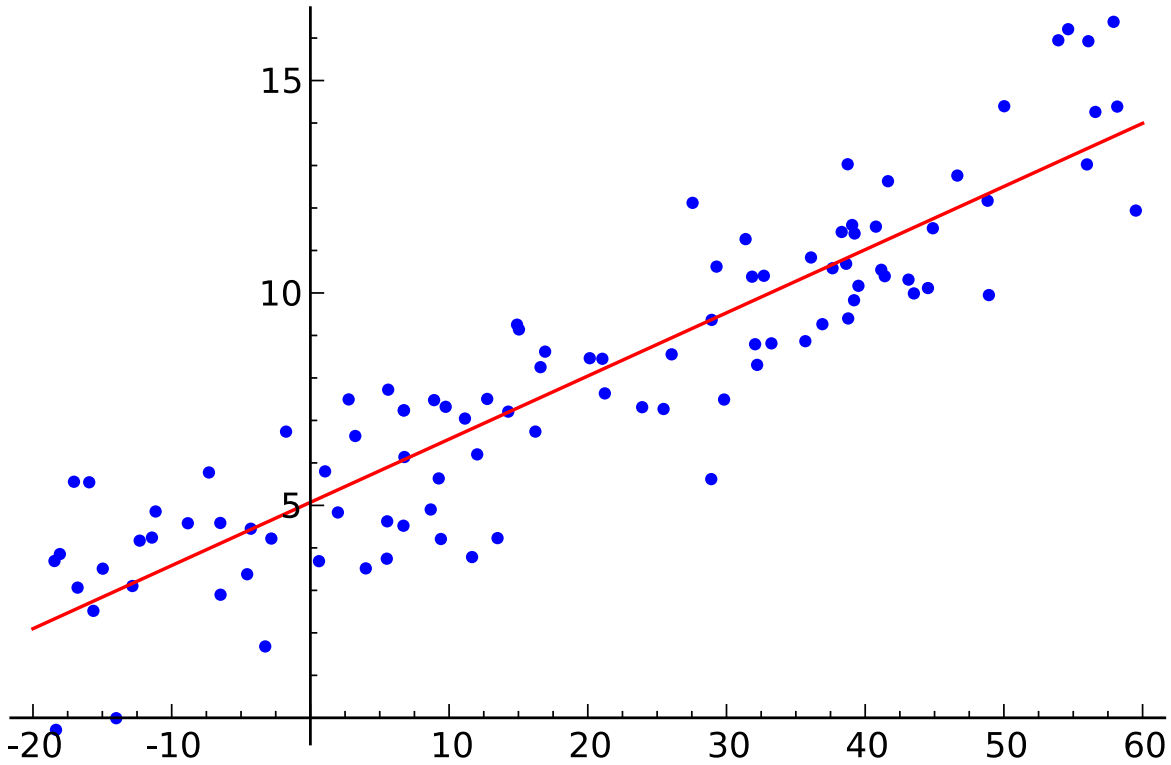
Separación en altas dimensiones

- Pasar de las features originales a features extendidas
  - Incrementar la dimensionalidad
- Por ejemplo, pasar de dos dimensiones x_1, x_2a seis dimensiones $1, \sqrt{2}x_1, \sqrt{2}x_2, x_1x_2, x_1^2, x_2^2$
- La función de expansión se llama **kernel**

Kernel Trick

- Hacer una optimización de programación cuadrática en gran cantidad de dimensiones sería muy costoso
- Pero en la optimización, si la función de expansión (kernel) tiene buenas propiedades es posible evitar la gran dimensionalidad y realizar todas las operaciones en la dimensión original
- SVMs son conceptualmente sencillas, la complejidad radica en lograr que esas ideas sencillas sean factibles de computar

Regresión Lineal



$$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathrm{T}} \beta + \varepsilon_i, \qquad i = 1, \ldots, n$$

Estimación por least-squares

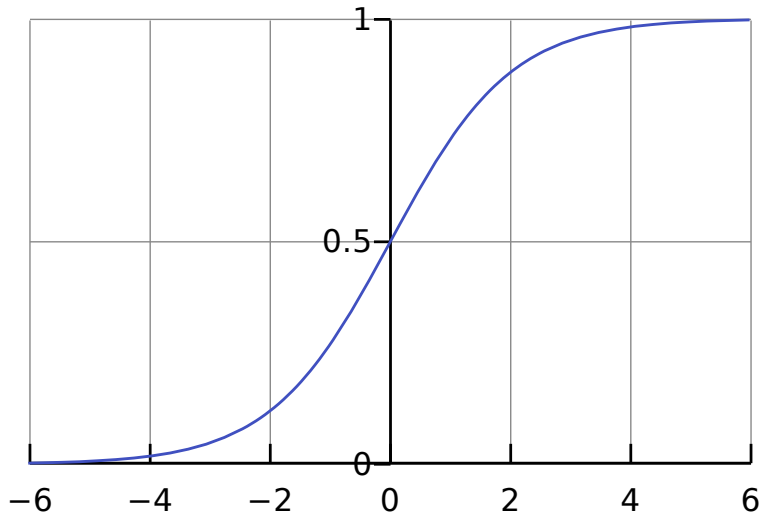
Representando en forma matricial $\mathbf{y} = \mathbf{X}\beta + \varepsilon$ donde

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

entonces podemos hacer una solución cerrada para β:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \left(\sum \mathbf{x}_i \mathbf{x}_i^T\right)^{-1}\left(\sum \mathbf{x}_i y_i\right)$$

Función Logística



$$F(t) = \frac{e^t}{e^t+1} = \frac{1}{1+e^{-t}}$$

Regresión Logística: intuición

- Pasar del espacio de la función objetivo al espacio de probabilidad de que la función objetivo sea de una clase determinada
- Minimizar el error de una combinación lineal de features después de aplicar la función logística para obtener una probabilidad
- No tiene solución cerrada, se utilizan métodos de tipo Newton para encontrar una solución aproximada

Distribuciones de la familia exponencial

- Una de las más versátiles
  - Incluyen Gaussianas, Poisson, Gamma, etc
- Llevan a problemas de optimización convexos (solubles computacionalmente)
- Describen la distribución de probabilidad como modelos lineales