HMM, CRF, CTC,...

Для чего

Классификация элементов последовательности в контексте, например:

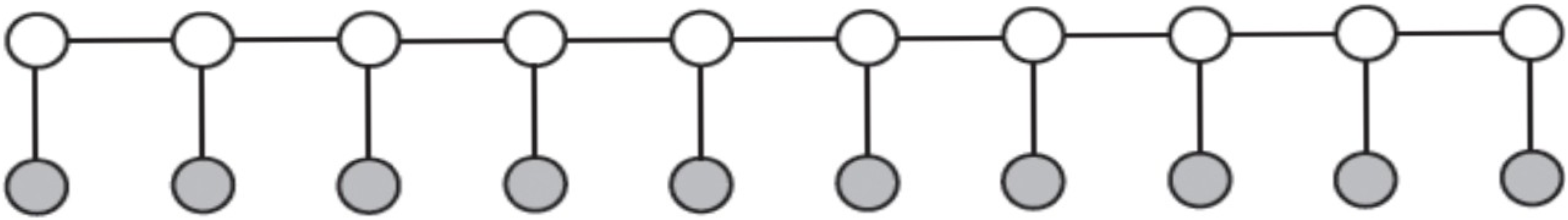
- POS tagging, NER
- OCR (при априорной сегментации)

Преобразование последовательности seq2seq:

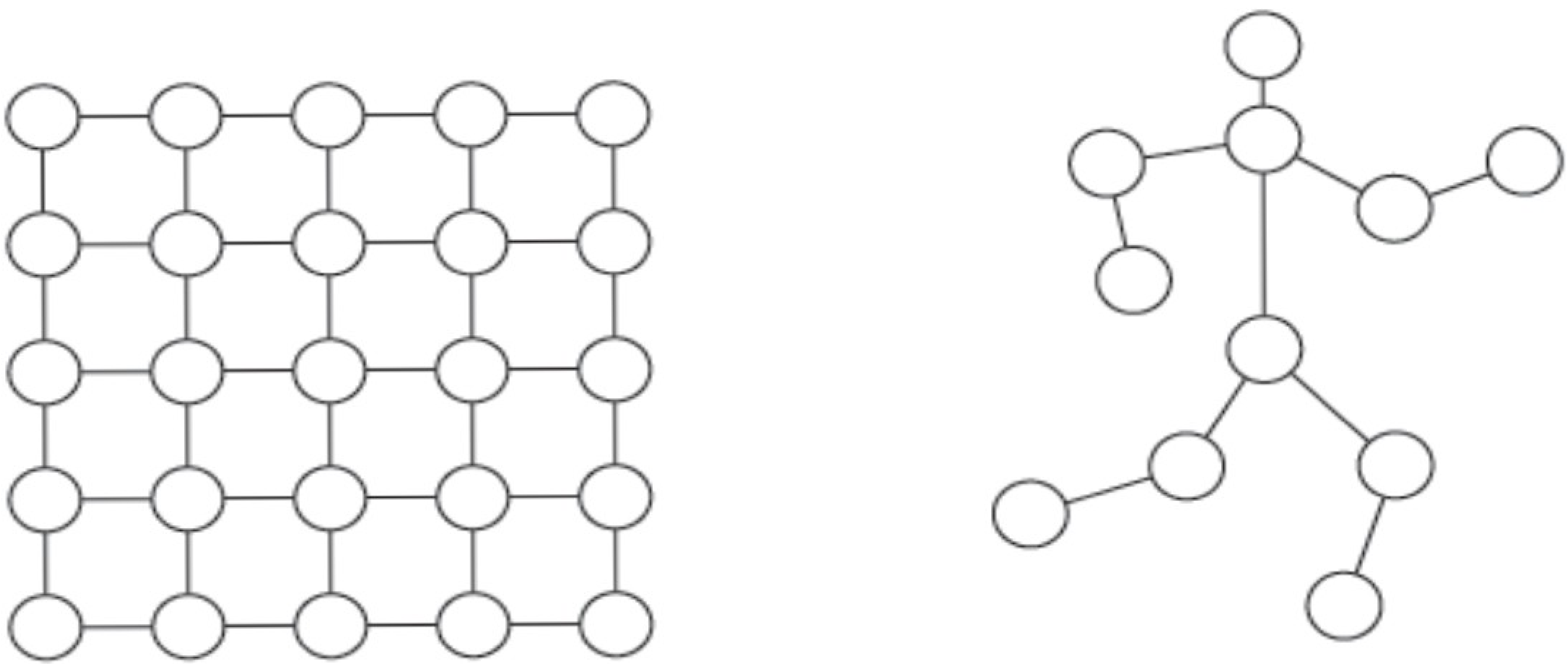
- Распознавание звука
- OCR

Цель хотим “сглаживать” результат классификации элементов по классам соседних элементов

Graphical models



Hidden Markov model



Graphical models

Байес

Дано: $\{(x, y)_i\}$

Нужна: $y = F(x)$

$F(x) = \text{argmax } p(y|x)$

Данные это совместное распределение $\{(x, y)_i\} \rightarrow p(x,y)$

Применим байеса:

$p(y|x) = p(x,y) / p(x) = p(x,y) / \sum_y p(x,y)$

Наивный Байес

Раскладываем (факторизуем) совместную вероятность

$p(x,y) \sim \prod_k p(x_k,y)$

Например

$p(x,A) = \prod_{kl} p(x_{kl}=\{0,1\},A)$

$\log p(x,A) = \sum_{kl} \log p(x_{kl}=\{0,1\},A)$

Получили растровый эталон (почти)

0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0
0	1	0	0	0	0	1	0
0	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1

Последовательность

Дано: $\{((x_1,y_1),\ldots,(x_n,y_n))_i\}$

Например: (мама, noun), (мыла, verb), (раму, noun)

$p(x, y) = \prod p(x_k,y_k|x_{k-1},y_{k-1}\ldots)$

“Наивно” упрощаем зависимости:

- $p(x, y) \sim \prod_k p(x_k|y_k)p(y_k|y_{k-1}\ldots)$

  Тогда:

  $p(y_k|y_{k-1}\ldots)$ - языковая модель

  $p(y_k|y_{k-1})$ – биграммная языковая модель – марковская модель

- Из обучающих данных мы знаем все Y, поэтому

  $p(y_k|y_{k-1}) = N(y_k y_{k-1}) / N(y_{k-1})$

Как использовать - Viterbi

$y = \text{argmax}\ \Pi_{k=1\ldots n} p(x_k|y_k)p(y_k|y_{k-1})\ \cancel{/ \sum_y p(x,y)}$

Надо перебрать $|y|^n$ комбинаций?

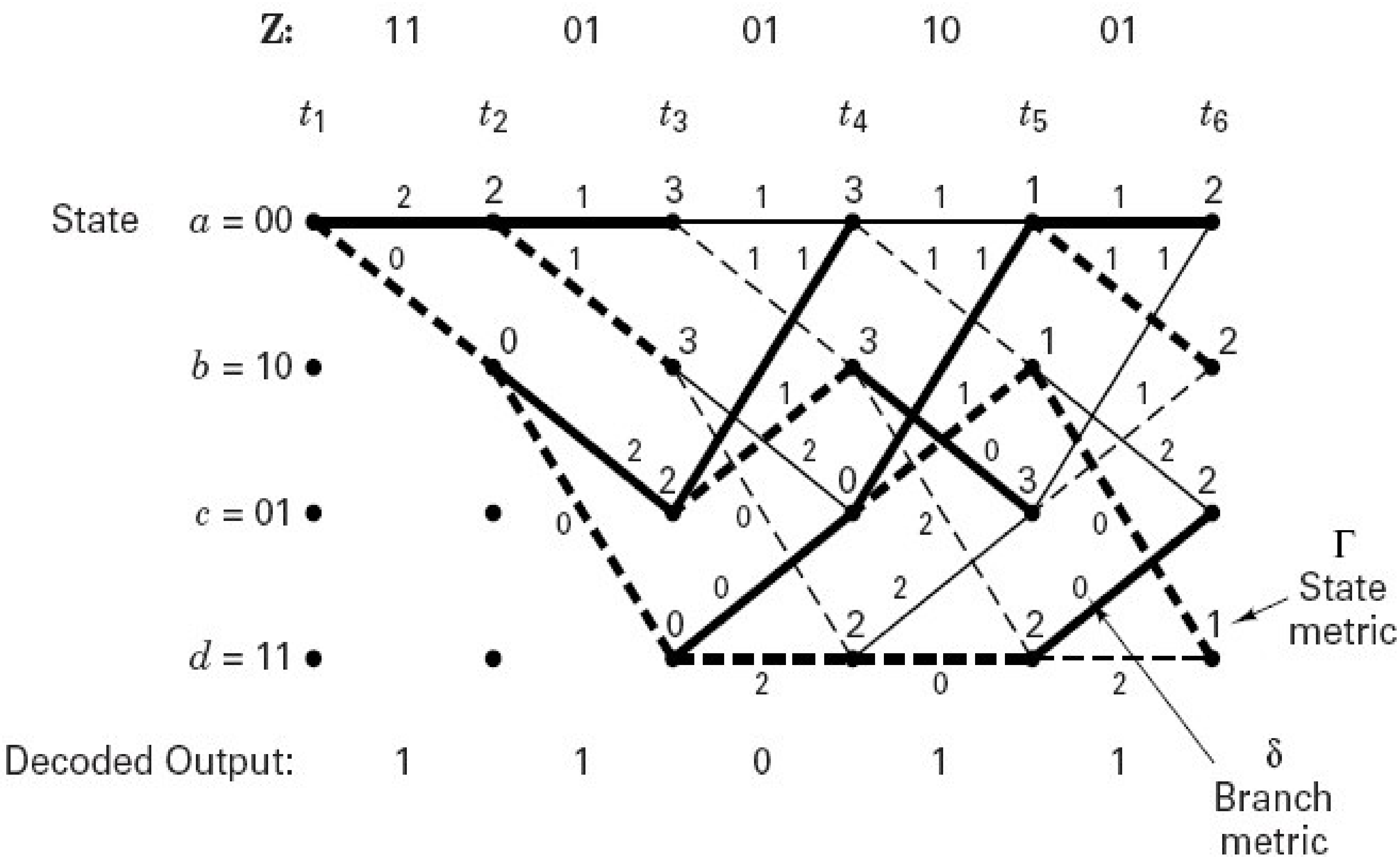
Динамическое программирование – для последовательности длины n-1 помним вероятность и перфикс пути для каждого варианта значения последнего элемента

$\max \Pi_{k=1\ldots n} p(x_k|y_k)p(y_k|y_{k-1}) = \max_{y_{n-1}}(p(x_n|y_n)p(y_n|y_{n-1})$

$\quad * \max \Pi_{k=1\ldots n-1} p(x_k|y_k)p(y_k|y_{k-1}))$

Шаг $y_k = \text{argmax}_{y_{k-1}}\ p(x_k|y_k)p(y_k|y_{k-1})p(y_{k-1}|\ldots)$

Viterbi


Z: 11 01 01 10 01
t_1 t_2 t_3 t_4 t_5 t_6
State
a = 00
b = 10
c = 01
d = 11
Γ
State
metric
δ
Branch
metric
Decoded Output: 1 1 0 1 1

Viterbi

Чем плох?

- Для N-gram надо комбинаторно размножать состояния на глубину N-1
- Например для триграмм символов ~ $50*50^2$ ~10000

HMM (простой)

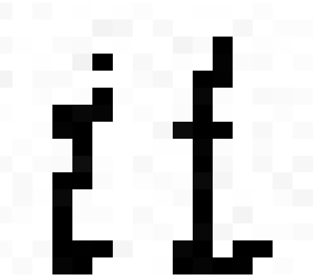
$\{((x_1,y_1),\ldots,(x_n,y_n))_i\}$

x – observations

- y – states
- $p(x, y) = \prod_k p(x_k|y_k)p(y_k|y_{k-1})$

- Мы уже знаем что это такое и как оно работает

Пример



p(y|x)

- p(1|i) = 0.5 p(i|i) = 0.5

  p(*L*|t) = 0.5 p(t|t) = 0.5

- p(y|y$_{-1}$)

  p(t|i) = 0.8 “it” p(t|1) = 0.05 “1t”

  p(L|i) = 0.05 “iL” p(L|1) = 0.05 “1L”

- p(“it”) = {p(i|i) = 0.5}*{p(t|t) = 0.5}*{p(t|i) = 0.8} = 0.2
- p(“1t”) = {p(1|i) = 0.5}*{p(t|t) = 0.5}*{p(t|1) = 0.05} = 0.0125

HMM(настоящий)

$\{(x_1 \ldots x_k, y_1 \ldots y_l)\}$

Т.е. входные данные и результат имеют разную "длину" (звук, OCR)

- Рассмотрим пример с мамой

"мама мыла раму" ↔ <noun> <verb> <noun>

x = {а-я\s}

y = {noun, verb, ...}

HMM

- Evaluation: $p(x)$
- Inference: argmax $p(y|x)$
- Training: $p(x|y)$ – emission, $p(y|y-1)$ – transition

HMM evaluation

$p(x|model) = \sum p(x|y)p(y) = \sum \prod p(x_i|y_i)p(y_i|y_{i-1})$

Можно переписать в рекурентную формулу

$p(x,y_{i\ldots}) = \sum_{i-1} p(x,y_{i-1})p(y_i|y_{i-1})p(x,y_{i-1\ldots})$

$alpha(i,y_i) = \sum_{i-1} p(x,y_{i-1})p(y_i|y_{i-1})alpha(i-1,y_{i-1})$

– где $alpha(i,y_i)$ – “forward probability”

Аналогично $beta(i,y_i)$ – “backward probability”

Вычисляется с помощью Витерби

HMM inference

$\text{argmax } p(y|x) = \text{argmax } p(x,y)$

$\max p(x,y_{i\ldots}) = \max_{i-1} p(x,y_{i-1})p(y_i|y_{i-1})(\max p(x,y_{i-1\ldots}))$

- Вычисляется с помощью Витерби – аналогично нграммам

HMM training

Найдем $p(y_i|y_j) = N(y_t=y_i, y_{t-1}=y_j) / N(y_{t-1}=y_j)$

Где взять N? У нас нет отображения t->y

-

  Вместо факта переход у нас $P(y_t=y_i, y_{t-1}=y_j, x)$ – запустим витерби для каждого обучающего примера

- $P(y_t=y_i, y_{t-1}=y_j, x) = alpha(j,t-1)p(x_t)p(y_i,y_j)beta(i,t)$

  где alpha(j,t-1) – вероятность префикса

  $alpha_{i,t} = \sum_k p(x_i|y_i)p(y_i|y_k)alpha_{k,t-1}$

  beta(i,t) – вероятность суффикса

HMM training

$P(y_t{=}y_i,\ y_{t-1}{=}y_j) = P(y_t{=}y_i,\ y_{t-1}{=}y_j,\ x) / P(x)$

$= P(y_t{=}y_i,\ y_{t-1}{=}y_j,\ x) / \text{alpha}(N)$

т.е. в каждом примере переход j → i случился

$\sum_t P(y_t{=}y_i,\ y_{t-1}{=}y_j)$ раз

Поэтому

$\underline{\mathbf{p}}(y_i,y_j) = N(y_t{=}y_i,\ y_{t-1}{=}y_j) / N(y_{t-1}{=}y_j)$

$= \sum_n \sum_t P(y_t{=}y_i,\ y_{t-1}{=}y_j) / \sum_n \sum_i \sum_t P(y_t{=}y_i,\ y_{t-1}{=}y_j)$

где $P(y_t{=}y_i,\ y_{t-1}{=}y_j) = \text{alpha}(j,t-1)p(x_t)p(y_i,y_j)\text{beta}(i,t) / \text{alpha}(N)$

где $\text{alpha}(N) = \sum_k p(x_i|y_i)\underline{\mathbf{p}}(y_i|y_k)\text{alpha}_{k,t-1}$

HMM

Научились вычислять $\underline{\mathbf{p}}(y_i,y_j)$ через $\underline{\mathbf{p}}(y_i,y_j)$!!!

Нужен EM алгоритм

- Инициализируем как-нибудь
- Вычисляем итеративно

MEMM

Maximum Entropy Markov Model

Мотивация:

– Не хотим считать вероятность $p(x,y)$
– Вместо x есть признаки – fx
  Нужна $p(y|fx,y_{-1}\ldots)$

MEMM

Позаимствуем рекурентную формулу

$p(y|fx,y_{-1}...) = p(y|fx,y_{i-1})p(y_{i-1...})$

$p(y|fx,y_{i-1})$ будем вычислять логистической регрессией

- $p(y|fx,y_{i-1}) = softmax(W*(fx,y_{i-1}))$

  т.е. просто учим классификатор переходить в нужное состояние

Почему Max Entropy

Пусть у нас есть набор сэмплов $x_1...x_n$

Мы хотим подобрать под них функцию распределения p(x)

Что мы про нее знаем

- это вероятность, поэтому $\sum p(x) = 1$
- она должна как-то соответствовать данным $x_1...x_n$- пусть хотя бы совпадает среднее $\sum p(x)x = \sum x_i/N$
- больше мы ничего не знаем – учтем это как то, что p(x) не должна далеко уходить от равномерного (“ничего не знаем”). Это измеряется KL расстоянием $\sum p \log(p/uniform)$

$\sum p \log(p/uniform) = \sum(p \log p – p \log uniform) = \sum p \log p – \log uniform \sum p = \sum p \log p – \log uniform$

Т.е. это $\sum p \log p + const = const – H(p)$ – значит мы хотим максимизировать энтропию

Почему softmax = Max Entropy

Теперь надо учесть два ограничения $\sum p(x) = 1$ и $\sum p(x)x = \sum x_i/N = M$

Применим метод Лагранжа

$L = \sum p \log p + \lambda_1(\sum p(x) - 1) + \lambda_2(\sum xp(x) - M)$

Найдем оптимум в пространстве функций

$\partial_p L = \partial_p(\sum p \log p + \lambda_1(\sum p(x) - 1) + \lambda_2(\sum xp(x) - M)) =$

$= \sum[\partial_p(p \log p) + \lambda_1 \partial_p p(x) + \lambda_2 \partial_{p(}(xp(x))] =$

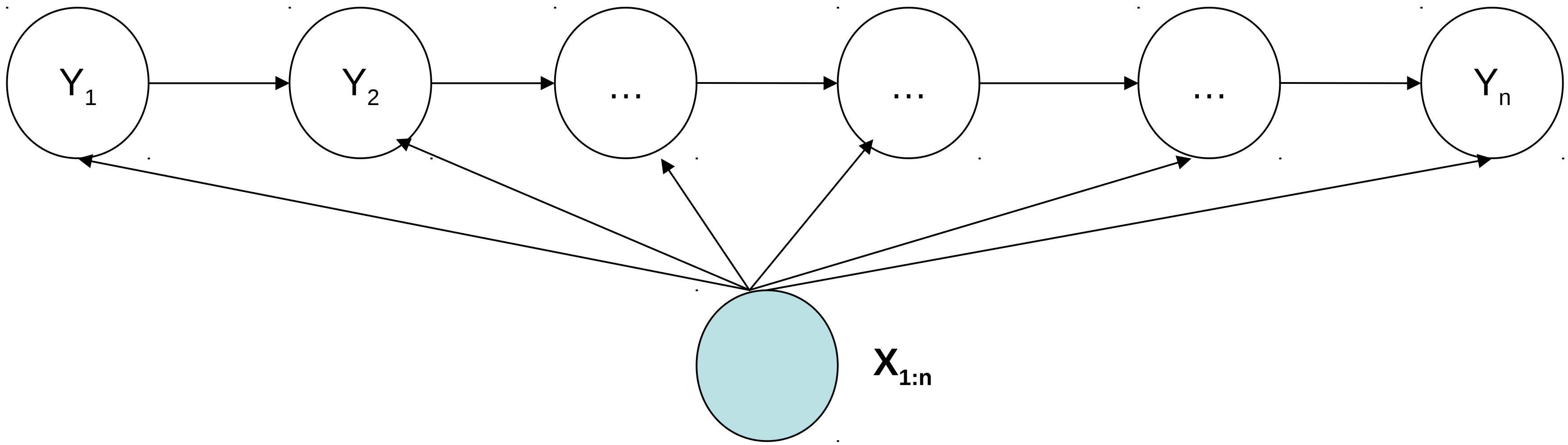
$= \sum[\log p + 1 + \lambda_1 \qquad + \lambda_2 x]$

В оптимуме $\partial_p L = 0$, поэтому $\sum[\log p + 1 + \lambda_1 + \lambda_2 x] = 0$

Отсюда получаем $p \sim \exp(x)$ – вот и логистическая регрессия

MEMM

Для p(y|...) можем использовать все что угодно



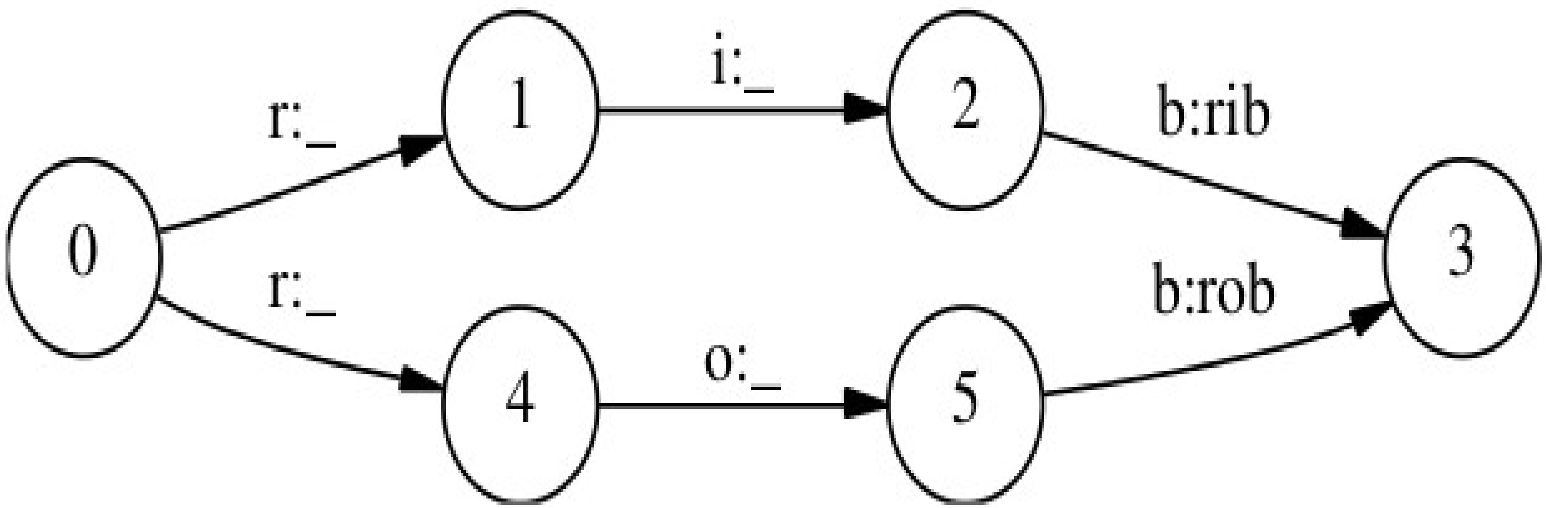
Не MEMM но тоже жадный

Вместо логистической регрессии можно использовать SVM-классификатор для жадного выбора состояния.

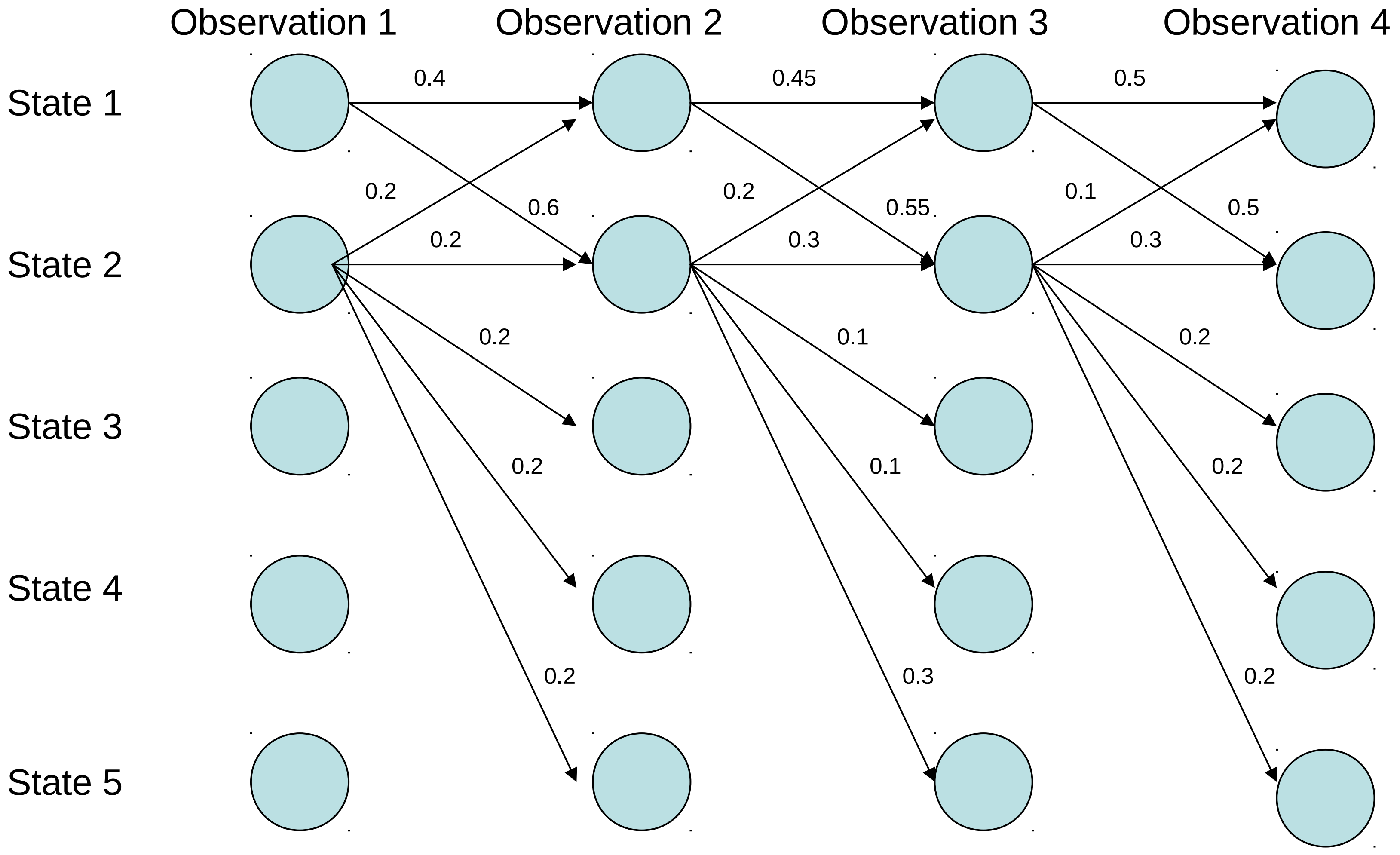
- Пример – MaltParser

MEMM

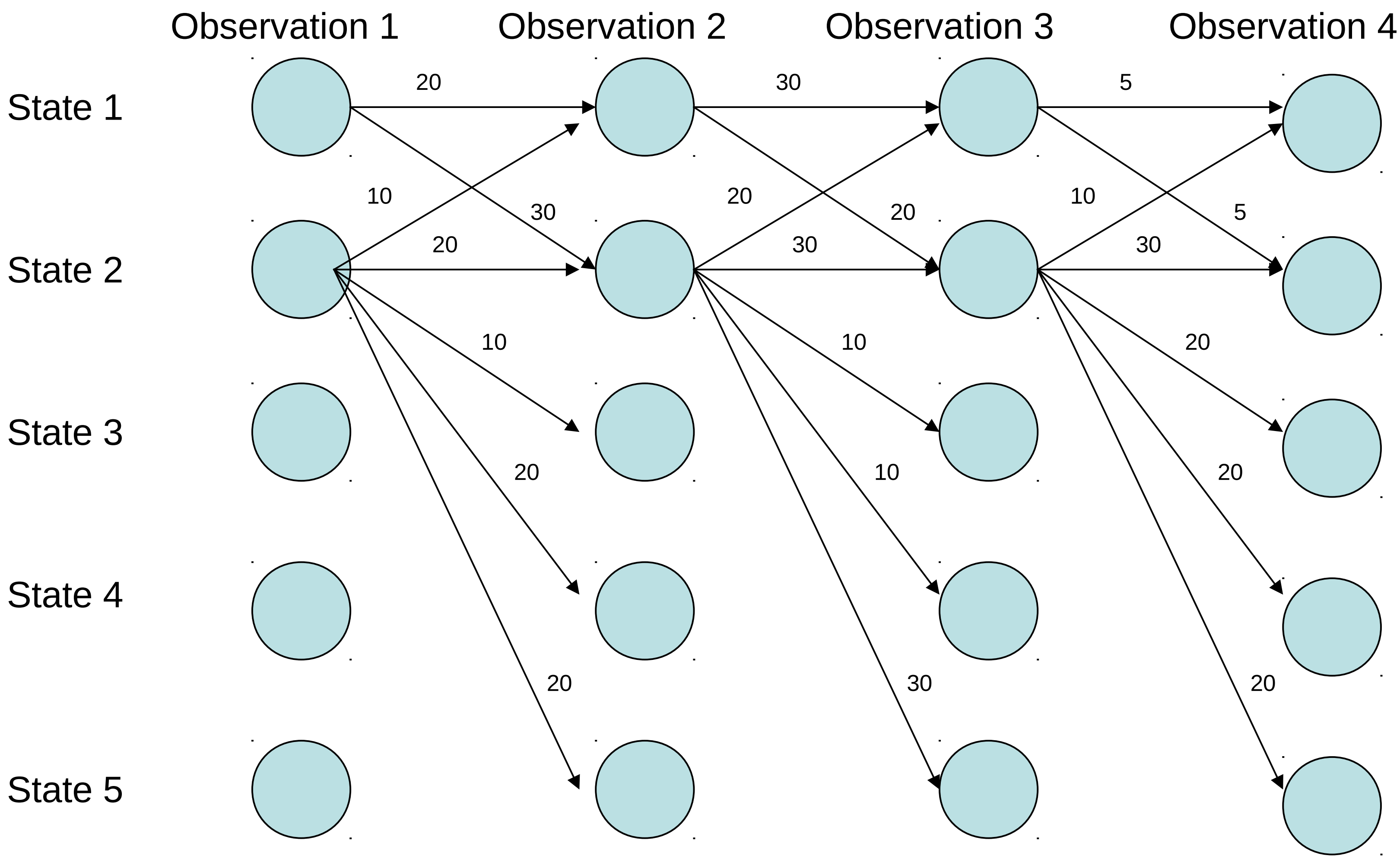
Плохо быть жадным или label bias problem



Label bias problem



Label bias problem



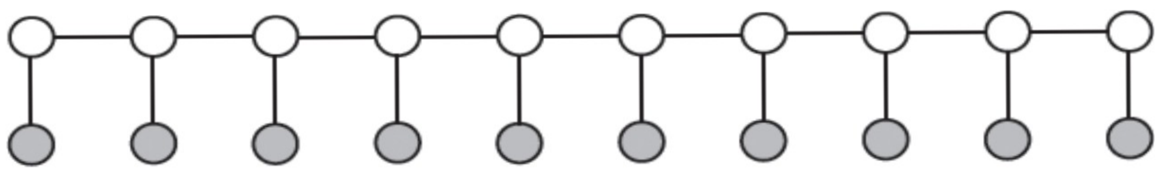
CRF

Conditional Random Field

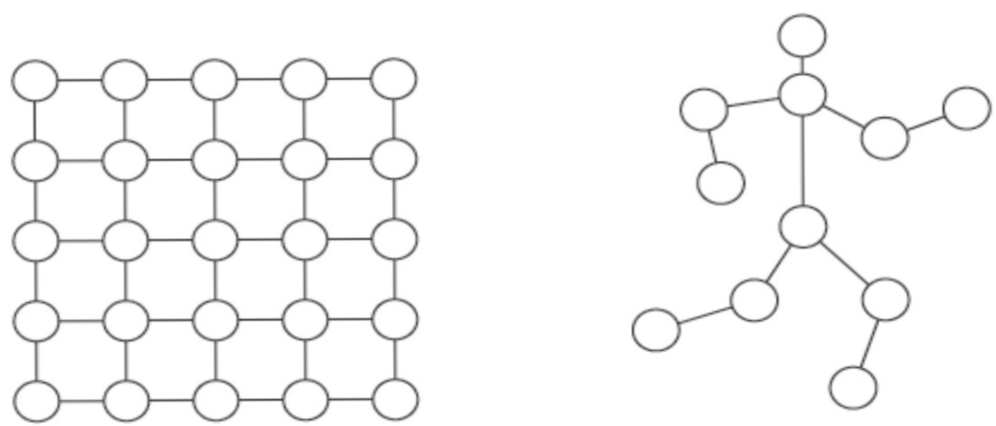
p(y|x,y[соседи в графе зависимостей])

Сегодня рассматриваем только цепочку

$p(y|x,y_{-1})$



Hidden Markov model



Graphical models

Linear chain CRF

Почти как MEMM

$p(y|x,y_{-1}) = \text{softmax}(W*[x,y_{-1}])$

$= \exp W*[x,y_{-1}] / \sum \exp W*[x,y_{-1}]$

Но выносим нормализацию на всю последовательность

$p(y|x) = \prod \exp W*[x_t,y_{t-1}] / Z$

– Где $Z = \sum_y \prod \exp W*[x_t,y_{t-1}]$

Немного log магии

Перейдем от p к logit

$\log(p) = \log(\prod \exp W*[x_t, y_{t-1}] / Z) =$

$= \log(\prod \exp W*[x_t, y_{t-1}]) - \log(Z) =$

$= \sum \log(\exp W*[x_t, y_{t-1}]) - \log(Z) =$

$= \sum W*[x_t, y_{t-1}] - \log(Z)$

CRF

- Inference: $y = \text{argmax}_y\ p(y|x)$
- Training: $W = \text{argmax}_w\ p(y|x)$

CRF

- Inference: $y = \text{argmax}_y\ p(y|x)$

  $= \text{argmax}_y \log p(y|x) =$

  $= \text{argmax}_y (\sum W^*[x_t, y_{t-1}] - \log(Z)) =$

  $= \text{argmax}_y \sum W^*[x_t, y_{t-1}]$

  Надо просто найти лучший путь с помощью витерби

CRF

Training: $W = \text{argmax}_w \; p(y|x)$

$= \text{argmax}_w \log p(y|x)$

Градиентный спуск: $\partial_W \log p(y|x)$

Дифференцируем крокодила

$\partial_W (\sum W^*[x_t, y_{t-1}] - \log(Z)) =$

$=[x_t, y_{t-1}] - \partial_W \log(Z)$

Дифференцируем Z-крокодила

$\partial_W \log(Z) = \partial_W Z \,/\, Z = \partial_W \sum_y \prod \exp W^*[x_t, y_{t-1}] \,/\, Z =$

$= \sum_y \prod \exp W^*[x_t, y_{t-1}]\ [x_t, y_{t-1}] \,/\, Z =$

$= P(y|x)[x_t, y_{t-1}]$

Нам нужно посчитать $P(y|x)$

Вспоминаем HMM – alpha, beta, и т.п.

CRF на практике

Есть разные варианты $W*[x_t,y_{t-1}]$

– $W_x x_{t\,,}$ $W_y y_{t-1}$ - классический

– $x_{t\,,}$ $W_y y_{t-1}$ – HMM-CRF

– x_t $W_y y_{t-1}$

+регуляризатор L2

∂_W $(\sum W*[x_t,y_{t-1}] – \log(Z) + \lambda W^2)$

можно ставить поверх нейросети

RNN+CRF=NER

```
blstm =  Bidirectional(LSTM(SENTENCE_LSTM_DIM,
    return_sequences=True))(embeddings)

blstm2 = Bidirectional(LSTM(50,
    return_sequences=True))(blstm)

result = CRF(n_out, sparse_target=True)

model = Model(input=[tokens_input, beginning_input,
    ending_input, casing_input], output=[result])
```

Немного физики

вася выписал   за   науку за   1913 г

PRS OUT   TIT TIT   OUT …

PRS OUT   OUT TIT   TIT   …

Поможет ли здесь CRF ?

Немного физики

Поможет если расщепить метки

вася выписал   за   науку за   1913 г

sPRS OUT   bTIT eTIT OUT …

sPRS_S OUT   OUT eTIT eTIT   …

CTC

Connectionist Temporal Classification

$\{(x_1 \ldots x_k, y_1 \ldots y_l)\}$

Пусть есть локальный классификатор

$p(y|x_i)$, который дает $y_1 \ldots y_k$

а есть $y_1 \ldots y_l$ где $l <= k$

CTC

Чтобы не путаться назовем *y* из классификатора *x*-ом

$p(y_k|x_k)$ – x_k

Складываем вероятности эквивалентных выравниваний

$p(y_{1..l}) = \sum_{x|y=B(x)} \prod_k p(x_k)$

B(x) – склейка повторяющихся дубликатов

- B(aaabb) = ab
- Чтобы уметь порождать aab, вводим сеператор _
- B(aa_abb) = ab

CTC

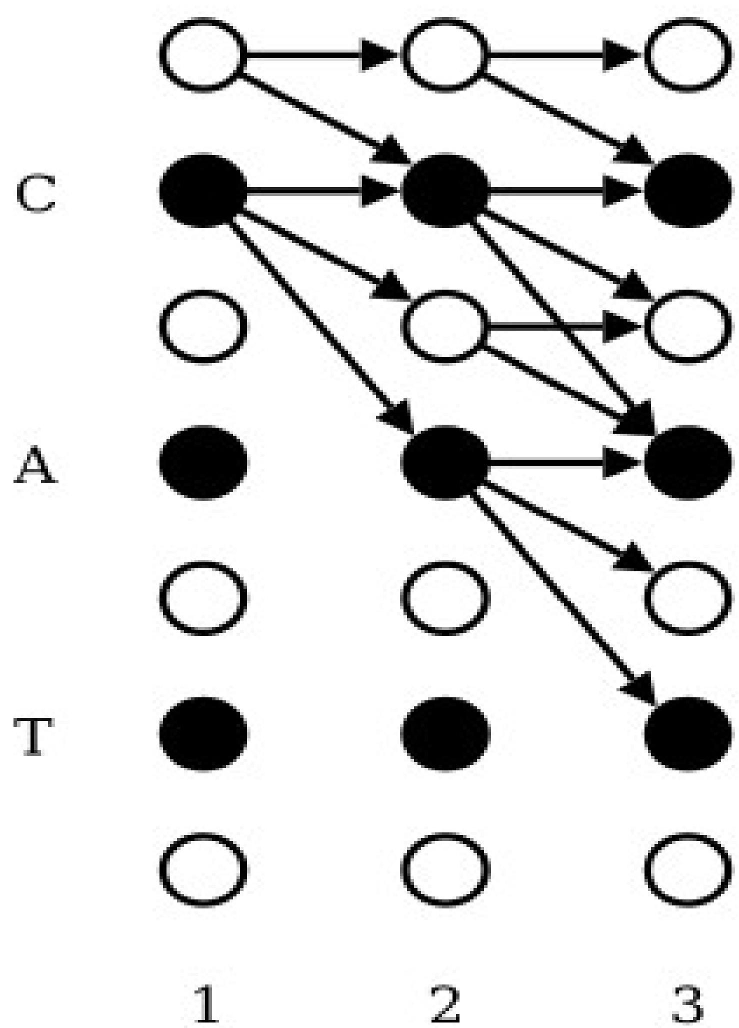
Учимся считать $p(y_{1..l}) = \sum_{x|y=B(x)} \prod_k p(x_k)$

Зовем на помощь альфу

$alpha(y_i,x_j) =$

- $p(x_j)alpha(i,x_{j-1}) \mid y_i = x_j$
- $p(x_j)(alpha(i,x_{j-1}) + alpha(i-1,x_{j-1})) \mid y_i \mathrel{!}= x_j$

CTC



CTC OCR

```
def build_model():
    model = Sequential()
    model.add(Input(shape=IMAGE_SIZE + (3,), dtype='float32'))
    model.add(conv_block(10, 3, 2))
    model.add(darknet_block(20, 40)
    model.add(darknet_block(20, 40)
    model.add(MaxPooling2D(pool_size=2, padding='same')
    model.add(darknet_block(30, 60)
    model.add(darknet_block(30, 60)
    model.add(MaxPooling2D(pool_size=2, padding='same')
    model.add(darknet_block(40, 80)
    model.add(darknet_block(40, 80)
    model.add(MaxPooling2D(pool_size=2, padding='same')(x)
    model.add(darknet_block(50, 100)
    model.add(darknet_block(50, 100)
    model.add(Permute((2,1,3))
    model.add(Reshape((IMAGE_SIZE[1] // 16, IMAGE_SIZE[0] // 16 * 50))
    model.add(Bidirectional(GRU(50, return_sequences=True))
    model.add(Dense(ALPHABET_SIZE))
    return model
```

CTC OCR

```
def ctc_lambda_func(args):
    return K.ctc_batch_cost(*args)

def build_ctc_model(ocr_model):
    labels = Input((PLATE_LEN,), dtype='float32')
    input_length = Input((1,), dtype='int64')
    label_length = Input((1,), dtype='int64')
    softmaxed = Activation('softmax')(ocr_model.output)
    loss_out = Lambda(ctc_lambda_func, output_shape=(1,))\
        ([softmaxed, labels, input_length, label_length])

    return Model(inputs=[ocr_model.input, labels, input_length, label_length],
        outputs=[loss_out])
```

RNN

- $y = RNN([x, y_{-1}])$

Задание

Написать Витерби на numpy

NUM_CLASSES=10

SEQ_LEN=100

x = np.random((SEQ_LEN,NUM_CLASSES))

T = np.random((NUM_CLASSES,NUM_CLASSES))

$y = x \, T \, y_{-1}$

y = ?