CSE440: NATURAL LANGUAGE PROCESSING II

Farig Sadeque
Associate Professor
Department of Computer Science and Engineering
BRAC University

Lecture 5: Sequence Learning

Outline

- Sequence tagging (SLP 8)
- Markov models (SLP Appendix A)
- Recurrent neural networks (SLP 9)

Sequences are common in languages

Fully furnished condo in the beautiful Catalina Foothills! Equipped with everything you need - house wares, linens, full-size washer & dryer, cable and Wi-Fi. Relax on your private covered patio or take a dip in the sparkling pool! Close to fine dining and shopping, too. List price is average, please call for exact pricing and availability.

Sequences are common in languages

Fully furnished condo in the beautiful Catalina Foothills!

WHAT LOCATION

Equipped with everything you need - house wares, linens,

full-size washer & dryer, cable and Wi-Fi. Relax on your

private covered patio or take a dip in the sparkling pool!

FEATURES

Close to fine dining and shopping, too. List price is

NEIGHBORHOOD

average, please call for exact pricing and availability.

PRICE CONTACT

Sequences are common in languages

Fully furnished condo in the beautiful Catalina Foothills!

Fully	furnished	condo	in	the	beautiful	Catalina	Foothills!
ADV	ADJ	NOUN	PREP	DET	ADJ	PROPN	PROPN

Sequences are common in languages

- Speech recognition
  - Group acoustic signal into phonemes
  - Group phonemes into words
- Natural language processing
  - Part of speech tagging
    - our running example
  - Named entity recognition
  - Information extraction
  - Question answering

Parts-of-speech tagging

Why not just make a big table?

- badger is a NOUN, trip is a VERB, etc.

Because part-of-speech changes with the surrounding sequence:

- I saw a badger in the zoo.
- Don’t badger me about it!
- I saw him trip on his shoelaces.
- She said her trip to Greece was amazing.

How big is this ambiguity issue?

Part-of-speech ambiguity

	WSJ		Brown	
Types:				
1 tag	44,432	(86%)	45,799	(85%)
2+ tags	7,025	(14%)	8,050	(15%)

Most words in the English vocabulary are unambiguous.

Part-of-speech ambiguity

	WSJ		Brown	
Types:				
1 tag	44,432	(86%)	45,799	(85%)
2+ tags	7,025	(14%)	8,050	(15%)
Tokens:				
1 tag	577,421	(45%)	384,349	(33%)
2+ tags	711,780	(55%)	786,646	(67%)

But, most words in running text are ambiguous! That is, ambiguous words are more prevalent.

A big table is still a good start

- Only 30-40% of words in running text are unambiguous.
- What if, we have a table for all words, and for ambiguous words, store the most commonly used tag for that word in there?
- This is called Most frequent tag baseline
    - assign each token the tag that it appeared with most frequently in the training data.
    - 92.34% accurate on WSJ corpus.

A big table is still a good start

- What’s the tag for *cut*?

  10 cut NN

  25 cut VB

  13 cut VBD

  7 cut VBN

Learning sequence taggers

- To improve over the most frequent tag baseline, we should take advantage of the sequence.
- Some options we will cover:
  - Hidden Markov models
    - Parameters estimated by counting (like naïve Bayes)
  - Maximum entropy Markov models
    - Parameters estimated by logistic regression
  - Recurrent neural networks

Hidden Markov Models

- Maximum entropy Markov models (MEMM)
- (Visible) Markov models for PoS tagging
- Training by counting
- Smoothing probabilities
- Handling unknown words
- Viterbi algorithm

Why POS Tagging Must Model Sequences

Our running example:

Secretariat is expected to race tomorrow.

Secretariat is ________

Race is ________

To understand context, we will predict all tags together.

Approach 0: Rule-based baseline

- Assign each word a list of potential POS labels using the dictionary
- Winnow down the list to a single POS label for each word using lists of hand-written disambiguation rules

Given input: "that"
if
(+1 A/ADV/QUANT); /* *if next word is adj, adverb, or quantifier* */
(+2 SENT-LIM); /* *and following which is a sentence boundary.* */
(NOT-1 SVOC/A); /* *and the previous word is not a verb like* */
/* *'consider' which allows adjs as object complements* */
then eliminate non-ADV tags
else eliminate ADV tag

You can learn these rules: see Transformation-based Learning: https://dl.acm.org/citation.cfm?id=218367

Approach 1: Maximum entropy Markov models

- Maximum entropy = logistic regression
- Markov models
  - Discovered by Andrey Markov
  - Limited horizon

А. А. Марков (1886).

- How would you implement sequence models in the logistic regression algorithm that we know?
- Let’s assume we scan the text left to right.

Approach 1 continued

- Add the previously seen tags as features!
  - Use gold tags in training
  - Use predicted tags in testing
- Other common features
  - Words, lemmas in a window [-k, +k]
  - Casing info, prefixes, suffixes of these words
  - Bigrams containing the current word

See also:
https://github.com/clulab/processors/blob/master/main/src/main/scala/org/clulab/processors/clu/sequences/PartOfSpeechTagger.scala

Approach 1: bidirectional MEMMs

- You can stack MEMMs that traverse the text in opposite directions:
    - Left-to-right direction (same as before)
    - Right-to-left: uses the prediction(s) of the above system as features!
        - What is the problem with the predictions of the left-to-right model here?
- Many state-of-the-art taggers use this approach: CoreNLP, processors, SVMTool

Approach 2: Hidden (visible) Markov Models

- Let's put the probability theory we covered in the previous lecture to use!
- The resulting approach is called (visible) Markov model
- "Visible" to distinguish it from the hidden Markov models, where the tags are unknown
    - Imagine implementing a POS tagger for an unstudied language without POS annotations

Approach 2: Hidden (visible) Markov Models

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n)$$

- Sentence 1 contains *n* words
- t_1^n - an assignment of POS tags to this sentence
- w_1^n - the words in this sentence
- $\hat{t}_1^n$ - the estimate of optimal tag assignment

Let’s formalize this

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

We have four probabilities: likelihood, prior, posterior and marginal likelihood.

- Prior: Probability distribution representing knowledge or uncertainty of a data object prior or before observing it
- Likelihood: The probability of falling under a specific category or class.
- Posterior: Conditional probability distribution representing what parameters are likely after observing the data object
- Marginal likelihood: likelihood function that has been integrated over the parameter space. Does not affect inference

Three Approximations

- Words are independent of the words around them
- Words depend only on their POS tags, not on the neighboring POS tags

$$P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i)$$

- A tag is dependent only on the previous tag

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1})$$

Replace in the original equation

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-1})$$

Word likelihoods

Tag transition probabilities

Computing Tag Transition Probabilities

In the Brown corpus (1M words)

- DT occurs 116,454 times
- DT is followed by NN 56,509 times

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

$$P(NN|DT) = \frac{C(DT, NN)}{C(DT)} = \frac{56,509}{116,454} = .49$$

Computing Word Likelihoods

In the Brown corpus (1M words)

- VBZ occurs 21,627 times
- VBZ is the tag for “is” 10,073 times

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

$$P(is|VBZ) = \frac{C(VBZ, is)}{C(VBZ)} = \frac{10,073}{21,627} = .47$$

Example

Secretariat/NNP is/BEZ expected/VBN to/TO **race**/VB tomorrow/NR

People/NNS continue/VB to/TO inquire/VB the/AT reason/NN for/IN the/AT **race**/NN for/IN outer/JJ space/NN

Let's see why VB is preferred in the first case

Example

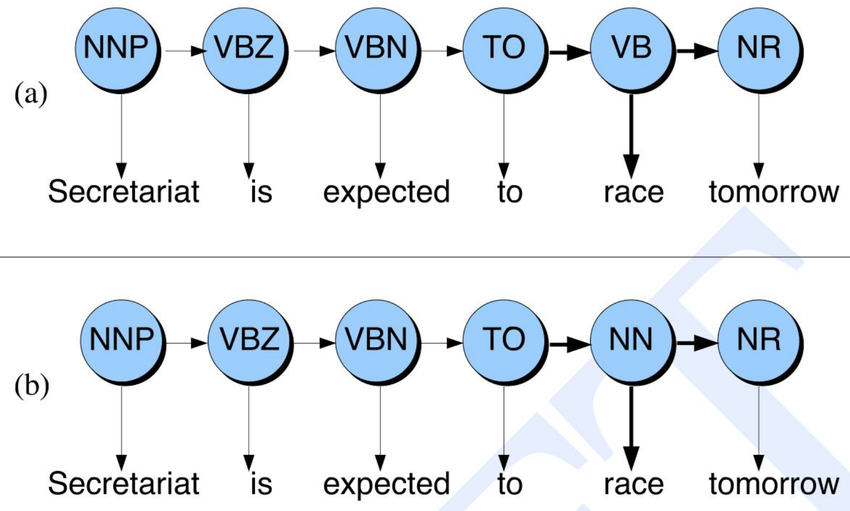


Figure 5.12 Two of the possible sequences of tags corresponding to the Secretariat sentence, one of them corresponding to the correct sequence, in which *race* is a VB. Each arc in these graphs would be associated with a probability. Note that the two graphs differ only in 3 arcs, hence in 3 probabilities.

Example

The first tag transition

- $P(NN|TO) = 0.00047$
- $P(VB|TO) = .83$

The word likelihood for “race”

- $P(race|NN) = 0.00057$
- $P(race|VB) = 0.00012$

The second tag transition

- $P(NR|VB) = 0.0027$
- $P(NR|NN) = 0.0012$

Example

$P(VB|TO)P(NR|VB)P(race|VB) = 0.00000027$

$P(NN|TO)P(NR|NN)P(race|NN) = 0.00000000032$

VB is more likely than NN, even though “race” appears more commonly as a noun!

Training/Testing an HMM

Just like with any machine learning algorithm, there are two important issues one needs to do to build an HMM:

- Training:
    - Estimating $p(t_i|t_{i-1})$ and $p(w_i|t_i)$
- Testing (predicting):
    - Estimating the best sequence of tags for a sentence (or sequence or words)

Training: Two Types of Probabilities

A: transition probabilities

- Used to compute the prior probabilities (probability of a tag)
- Often called tag transition probabilities

B: observation likelihoods

- Used to compute the likelihood probabilities (probability of a word given tag)
- Often called word likelihoods

Testing: Viterbi Algorithm

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-1})$$

Viterbi algorithm

- Computes the argmax efficiently
- Example of dynamic programming

What is a viterbi?

Andrew Viterbi

Engineer

Andrew James Viterbi is an American electrical engineer and businessman who co-founded Qualcomm Inc. and invented the Viterbi algorithm. Wikipedia

Born: March 9, 1935 (age 78), Bergamo, Italy

Books: CDMA, Principles of digital communication and coding

Education: University of Southern California (1963), More

Awards: IEEE Medal of Honor, Claude E. Shannon Award, More

Illustration of Search Space

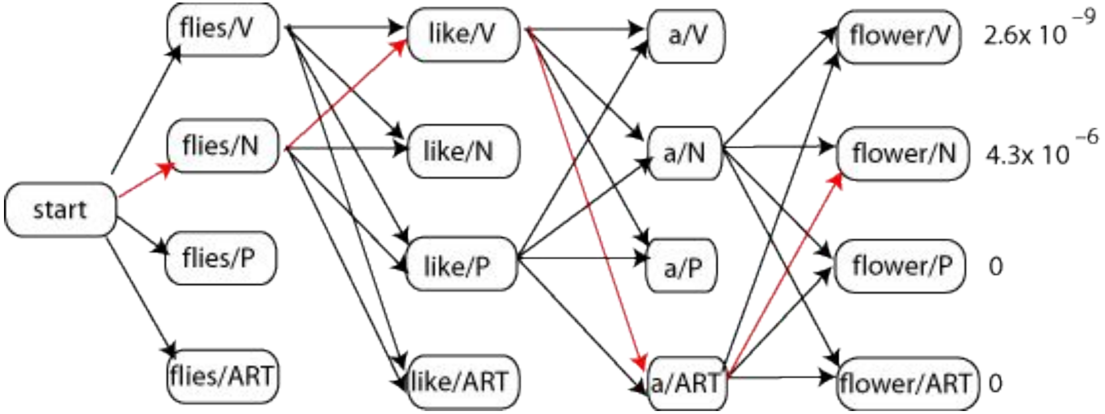
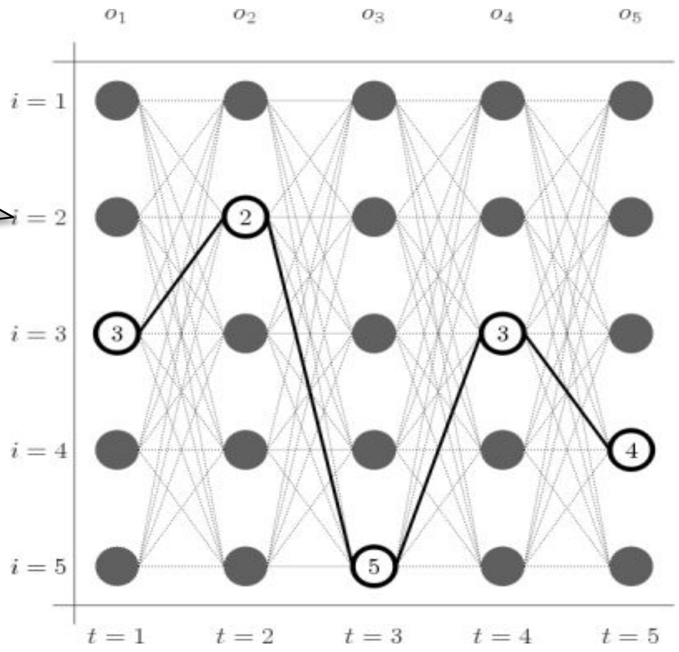


Illustration of Search Space



One row for each state (tag)

This is called a trellis

One column for each observation (word)

Viterbi Algorithm

Input

- State (or tag) transition probabilities (A)
- Observation (or word) likelihoods (B)
- An observation sequence O

Output

- Most probable state sequence Q together with its probability

Both A and B are matrices with probabilities

Example of A and B matrices

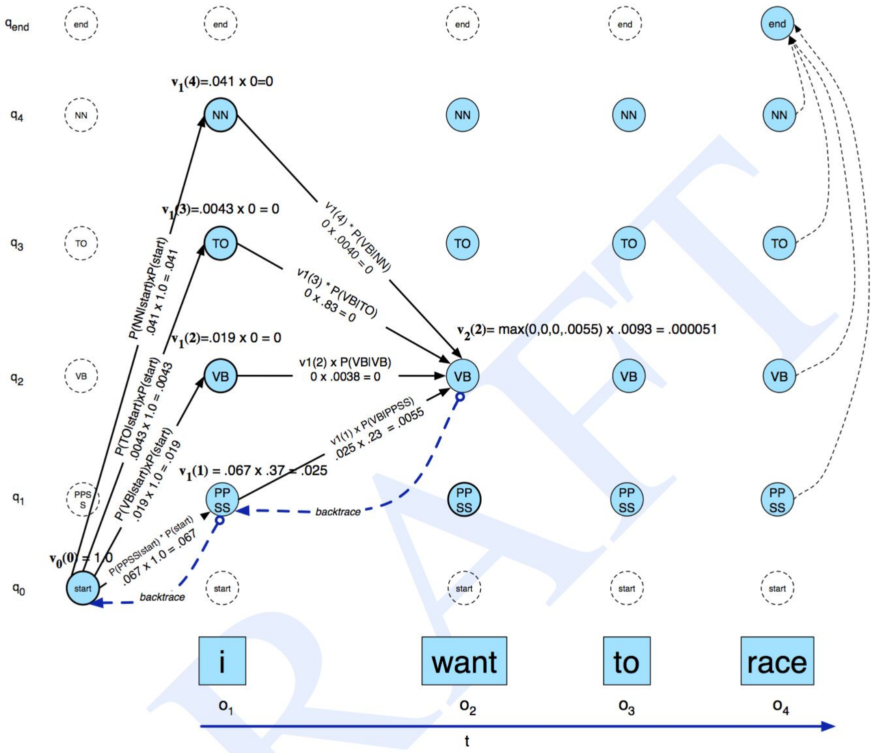
A: The rows are labeled with the conditioning event, e.g., P(PPSS|VB) = .0070

	VB	TO	NN	PPSS
<s>	.019	.0043	.041	.067
VB	.0038	.035	.047	.0070
TO	.83	0	.00047	0
NN	.0040	.016	.087	.0045
PPSS	.23	.00079	.0012	.00014

B: same as A, rows: conditioning events, e.g. P(want|NN) = .000054

	I	want	to	race
VB	0	.0093	0	.00012
TO	0	0	.99	0
NN	0	.000054	0	.00057
PPSS	.37	0	0	0

Example Trace



Summary of Viterbi Algorithm

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

- $v_{t-1}(i)$ – the **previous Viterbi path probability** from the previous time step t – 1 (i.e., the previous word)
- a_{ij} – the **transition probability** from previous state q_i (i.e., the previous word having POS tag i) to current state q_j (i.e., the current word having POS tag j)
- $b_j(o_t)$ – the **state observation likelihood** of the observation symbol o_t (i.e., word at position t) given the current state j (i.e., the j POS tag)

Extending the HMM Algorithm to Trigrams

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

This is pretty limiting for POS tagging
Let's extend it to trigrams of tags!

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \left[\prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1},t_{i-2})\right] P(t_{n+1}|t_n)$$

This is better

- t_{n+1} – end of sentence tag
- We also need virtual tags, t_0 and t_{-1}, to be set to the beginning of sentence value.

TnT

- This is what the TnT (Trigrams’n’Tags) tagger does
- Probably the fastest POS tagger in the world
- Not the best, but pretty close (96% acc)
- http://www.coli.uni-saarland.de/~thorsten/tnt/

Problems with TnT

$$P(t_i|t_{i-1},t_{i-2}) = \frac{C(t_{i-2},t_{i-1},t_i)}{C(t_{i-2},t_{i-1})}$$

Very sparse!

Backoff model: linear interpolation

$$P(t_i|t_{i-1}t_{i-2}) = \lambda_3 \acute{P}(t_i|t_{i-1}t_{i-2}) + \lambda_2 \acute{P}(t_i|t_{i-1}) + \lambda_1 \acute{P}(t_i)$$

λ1 + λ2 + λ3 = 1, to guarantee that result is a probability.

Deleted Interpolation

function DELETED-INTERPOLATION(*corpus*) **returns** $\lambda_1, \lambda_2, \lambda_3$

$\lambda_1 \leftarrow 0$
$\lambda_2 \leftarrow 0$
$\lambda_3 \leftarrow 0$
foreach trigram t_1, t_2, t_3 with $f(t_1, t_2, t_3) > 0$
  depending on the maximum of the following three values
    case $\frac{C(t_1,t_2,t_3)-1}{C(t_1,t_2)-1}$: increment λ_3 by $C(t_1, t_2, t_3)$
    case $\frac{C(t_2,t_3)-1}{C(t_2)-1}$: increment λ_2 by $C(t_1, t_2, t_3)$
    case $\frac{C(t_3)-1}{N-1}$: increment λ_1 by $C(t_1, t_2, t_3)$
  end
end
normalize $\lambda_1, \lambda_2, \lambda_3$
return $\lambda_1, \lambda_2, \lambda_3$

Other Types of Smoothing

- Add one:
  - $$P(w \mid t) = \frac{C(w,t)+1}{C(t)+K}$$
  - Where K is the number of words with POS tag t
- Variant of add one (Charniak's):
  - $$P(t_i \mid t_{i-1}) = (1-\varepsilon)\frac{C(t_i, t_{i-1})}{C(t_{i-1})} + \varepsilon$$
  - Not a proper probability distribution!

Another Problem for All HMMs

- Massive multiplication here:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-1})$$

Yet Another Problem: Unknown Words

- Solution 0 (not great): assume uniform emission probabilities (this is what "add one" smoothing does)
  - You can exclude closed-class POS tags such as…
  - This does not use any lexical information such as suffixes
- Solution 1: capture lexical information:

$$P(w^l|t^j) = \frac{1}{Z} P(\text{unknown word}|t^j) P(\text{capitalized}|t^j) P(\text{endings/hyph}|t^j)$$

  - This reduces error rate for unknown words from 40% to 20%

Main Disadvantage of HMMs

Hard to add features in the model

- Capitalization, hyphenated, suffixes, etc.

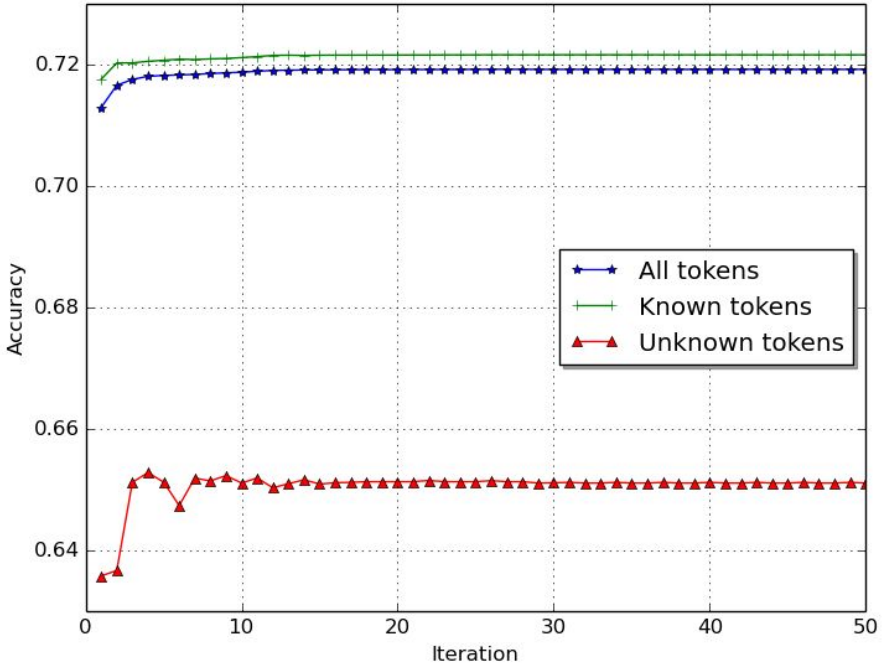
It's possible but every such feature must be encoded in the p(word|tag)

- Redesign the model for every feature!
- MEMMs avoid this limitation, but they take longer to train

Evaluation

- POS tagging accuracy = 100 x (number of correct tags) / (number of words in dataset)
- Accuracy numbers currently reported for POS tagging are most often between 95% and 97%
- But they are much worse for “unknown” words

Evaluation example



Evaluation

- Accuracy does not work. Why?
- We need precision, recall, F1:
    - $P = TP/(TP + FP)$
    - $R = TP/(TP + FN)$
    - $F1 = 2PR/(P + R)$
- Micro vs. macro F1 measures