Introduction to Conditional Random Fields (CRFs)

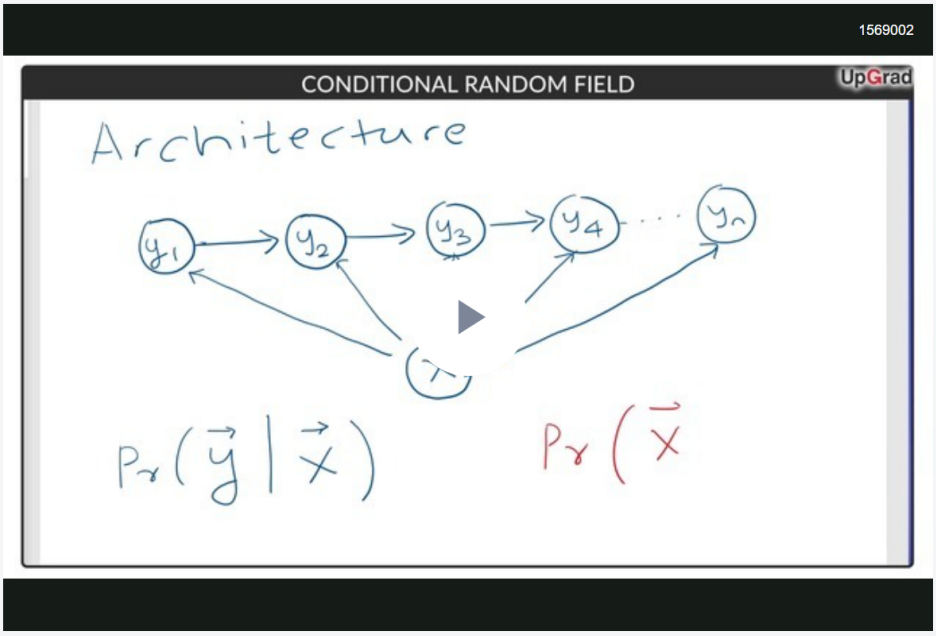
You have already learnt some probabilistic models used for sequence prediction tasks - unigram and bigram models, conventional ML models (naive Bayes. decision trees etc.), and HMMs.

You will now study another family of models commonly used for sequence prediction tasks - **Conditional Random Fields (CRFs).**

CRFs are used in a wide variety of sequence labelling tasks across various domains - POS tagging, speech recognition, NER, and even in computational biology for modelling genetic patterns etc. In this section, you will learn the architecture of CRFs in the context of entity recognition, though the same can be applied any of the tasks mentioned above.

CRFs are commonly used as an alternative to HMMs and, in some applications, have empirically proven to be significantly more accurate than HMMs.

CRFs are **discriminative probabilistic classifiers** (often represented as undirected graphical models in some texts). In the following lecture, you will study the distinction between discriminative and generative classifiers as well.



Broadly speaking, there are two types of classifiers in ML:

- **Discriminative classifiers** learn the boundary between classes by modelling the **conditional**

1. **Discriminative classifiers** learn the boundary between classes by modelling the **conditional probability distribution P(y|x)**, where y is the vector of class labels and x represents the input features. Examples are Logistic Regression, SVMs etc.

2. **Generative classifiers** model the **joint probability distribution P(x,y)**. Examples of generative classifiers are Naive Bayes, HMMs etc.

To summarise, CRFs model the conditional probability P(Y|X), where Y is the vector of output sequence (IOB labels here) and X is the input sequence (words to be tagged).

Ashish also mentioned that rather than feeding the input sequence X as it is, we usually **derive features from the input sequence** and feed the features. For now, just remember this sentence - you will shortly see what kind of 'features' CRFs generally use.

Additional Reading

- A concise explanation of the difference between discriminative and generative models


Report an error

NEXT
CRF Model Architecture

CRF Model Architecture

In this segment, you'll study the CRF model architecture in detail. Let's start with CRFs' '**feature functions**'.

CRF Feature Functions

We had mentioned in the previous segment that CRFs use 'feature functions' rather than the input word sequence **x** itself. The idea is similar to how we had extracted features for building the naive Bayes and decision tree classifiers in a previous section. Some example 'word-features' (each word has these features) are:

- Word and POS tag based features: word_is_city, word_is_digit, pos, previous_pos, etc.
- Label-based features: previous_label

Let's spend some time to understand CRF features in detail.

Consider a sequence of n tokens $x = (x_1, x_2, x_3, \ldots x_n)$ and the corresponding IOB label sequence $y = (y_1, y_2, y_3, \ldots y_n)$. The **task** is to predict the label sequence y using the word sequence x.

Similar to conventional classifiers, we define some feature functions in CRFs which we'll denote as f. A feature function takes the following **four inputs**:

1. The input sequence of words: x
2. The position i of a word in the sentence (whose features are to be extracted)
3. The label y_i of the current word (the target label)
4. The label y_{i-1} of the previous word

There are usually multiple feature functions, let's denote the j^{th} feature function as $f_j(x, i, y_i, y_{i-1})$.

We restrict the model to extract label-based features using only the previous label (and not using labels of the previous n words where $n > 1$). This assumption is similar to the first-order Markov assumption we had used in POS tagging (i.e. transition probabilities depend only on the previous POS tag). Also, the feature functions return a real-valued number, which is often 0 or 1 (i.e. binary output).

Let's take an example of a feature function f_1 which returns **1** if the word x_i is a city **and** the corresponding label y_i is 'I-location', else 0. This can be represented as:

$f_1(x, i, y_i, y_{i-1})$ =[[x_i is in city list name] & [y_i is I-location]]

The feature function returns 1 only if both the conditions are satisfied, i.e. when the word is a city name and is tagged as 'I-location' (e.g. Chicago/I-location).

Similarly, a second feature function can be defined using the previous label, for e.g.:
$f_2(x, i, y_i, y_{i-1})$ = [[y_{i-1} is B-location] & [y_i is I-location]]

You can define multiple such feature functions f_i, though all are not equally important. Every feature function f_i has a **weight** w_i associated with it, which represents the 'importance' of that feature function. This is almost exactly the same as logistic regression where coefficients of features represent their importance.

Training a CRF means to **compute the optimal weight vector** w which best represents the observed sequences y for the given word sequences x. In other words, we want to find the set of weights w which maximises $P(y|x, w)$. This is similar to training a logistic regression model to find the optimal coefficients by maximising $P(y|x, w)$ where w represents the vector of logistic regression model coefficients. You'll study CRF model training shortly, for now, let's focus on the model architecture itself.

Recall that in logistic regression, the conditional probability $P(y|x, w)$ is modelled using a sigmoid function.

In CRFs, we model the conditional probabilities $P(y|x, w)$ in a similar fashion. If there are k feature functions (and thus k weights), for each word i in the sequence x, we define a **scoring function** as follows:

$$score_i = exp(w_1.\, f_1 + w_2.\, f_2 \ldots + w_k.\, f_k) = exp(w.\, f(y_i, x_i, y_{i-1}, i))$$

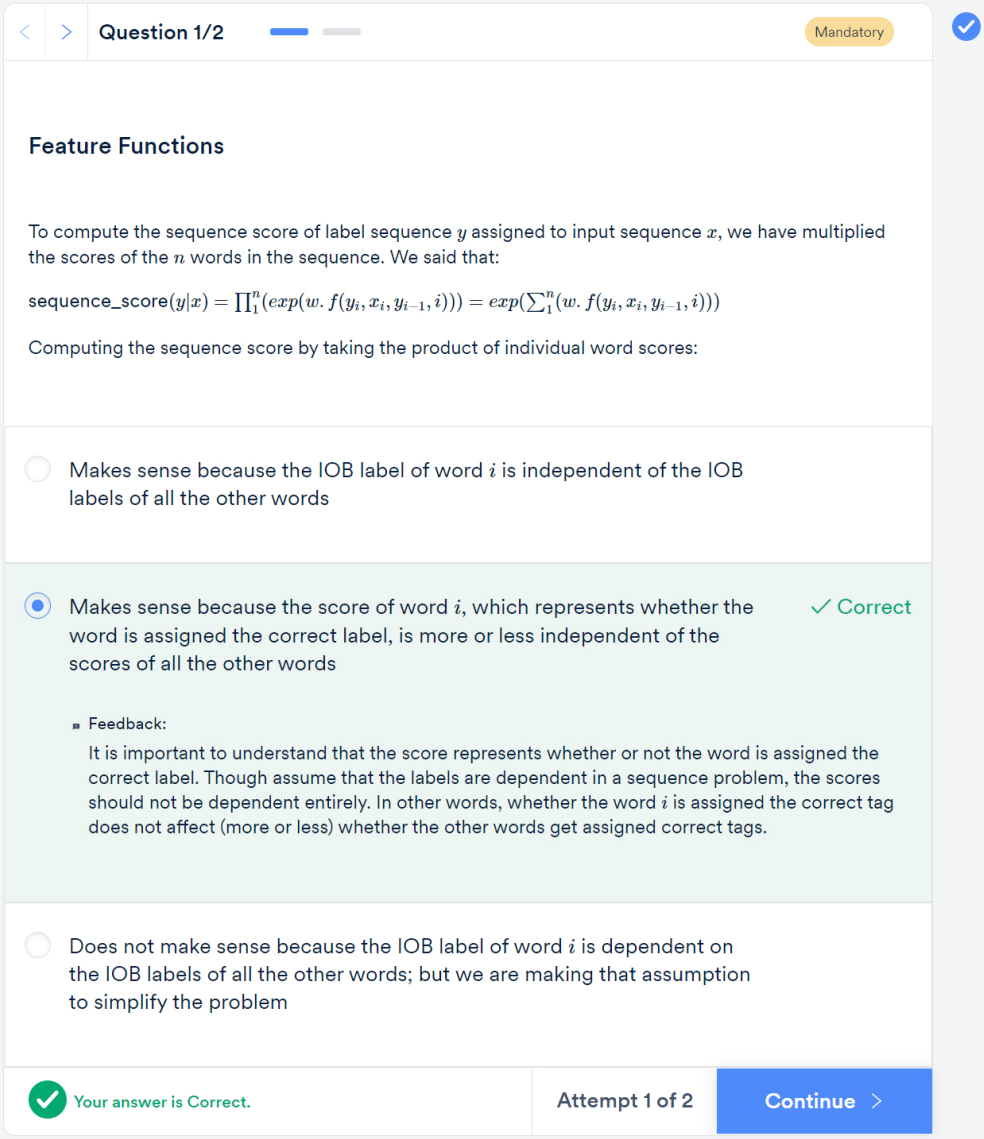
The intuition for **the score of the i^{th} word** is that it returns a high value if the correct tag is assigned to the word, else a low value. For e.g. say you want to tag the word 'Francisco' in the sentence 'The flight to San Francisco.' Consider that you have only two feature functions f_1 and f_2 defined above. Now, if you assign the (correct) label 'I-location' to 'Francisco', both f_1 and f_2 return a 1, and they return a 0 for any other tag such as 'O'. Also, $exp(x)$ being a monotonically increasing function, it returns a high value for high values of x. Thus, the score of a word is high when the correct tag is assigned to a word.

The score above is defined for each i^{th} word in the sequence x. If the sequences x and y are of length n, then the score of the sequence y is given by the **product of scores** of individual words in x. The sequence score will be highest when all the words have been assigned the correct labels:

$$\textbf{sequence\_score}(y|x) = \prod_{i=1}^{n}(exp(w.\, f(y_i, x_i, y_{i-1}, i)))$$

The product of exponents can be expressed as the exponential of the sums as follows:

$$\text{sequence\_score}(y|x) = \prod_{i=1}^{n}(exp(w.\, f(y_i, x_i, y_{i-1}, i))) = exp(\sum_{1}^{n}(w.\, f(y_i, x_i, y_{i-1}, i)))$$



Scores to Probabilities

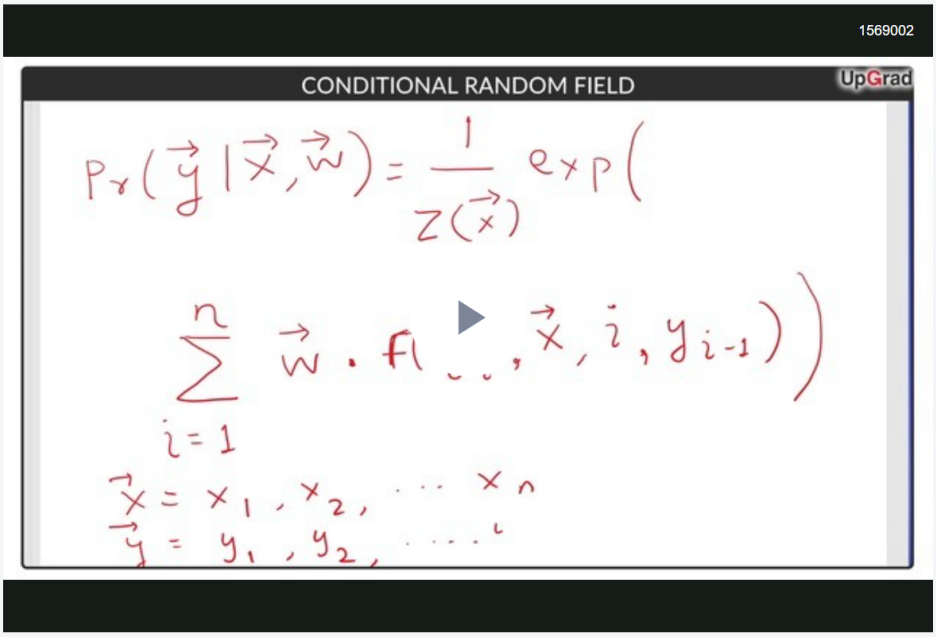
Using some math, we can now translate the scores of label sequences, which can be any real numbers, to probabilities. The probability $P(y|x, w)$, i.e. the probability of observing the sequence y given x for a certain w, is given by the score of the sequence y divided by the total score of all possible label sequences. If you have n words and t IOB labels, there are t^n possible IOB sequences y. The **inference task** is to find the optimal sequence y.

The denominator in the following expression, $Z(x)$, is a **normalising constant** which represents the sum of scores of all possible label sequences of x:

$$P(y|x, w) = exp(\sum_{1}^{n}(w.\, f(y_i, x_i, y_{i-1}, i)))/Z(x) = exp(w.\, f(x, y))/Z(x)$$

= score of sequence y assigned to x / sum of scores of all possible sequences

The term $exp(w.f(x,y))$ is a concise form of the expression $exp(\sum_1^n(w.f(y_i,x_i,y_{i-1},i)))$ which represents the score of the sequence y assigned to x. Let's have Ashish explain this further.



To summarise, the probability of observing the label sequence y given the input sequence x is given by:

$$P(y|x,w) = exp(\sum_1^n(w.f(y_i,x_i,y_{i-1},i)))/Z(x) = exp(w.f(x,y))/Z(x)$$

= score of sequence y assigned to x / sum of scores of all possible sequences

For now, assume that you have somehow computed the optimal weights w by training the model - you'll study that in the next segment. For a sentence of n words and t IOB labels, there are t^n possible label sequences, and your task is to find the optimal sequence y_{opt} . You will study how to do this in a later segment.

Let's now understand the mathematical form of the normalising constant $Z(x)$ and some examples of feature functions. You have already seen that $Z(x)$ is the sum of scores of all possible label sequences. Since the probability of a tag sequence y is given by

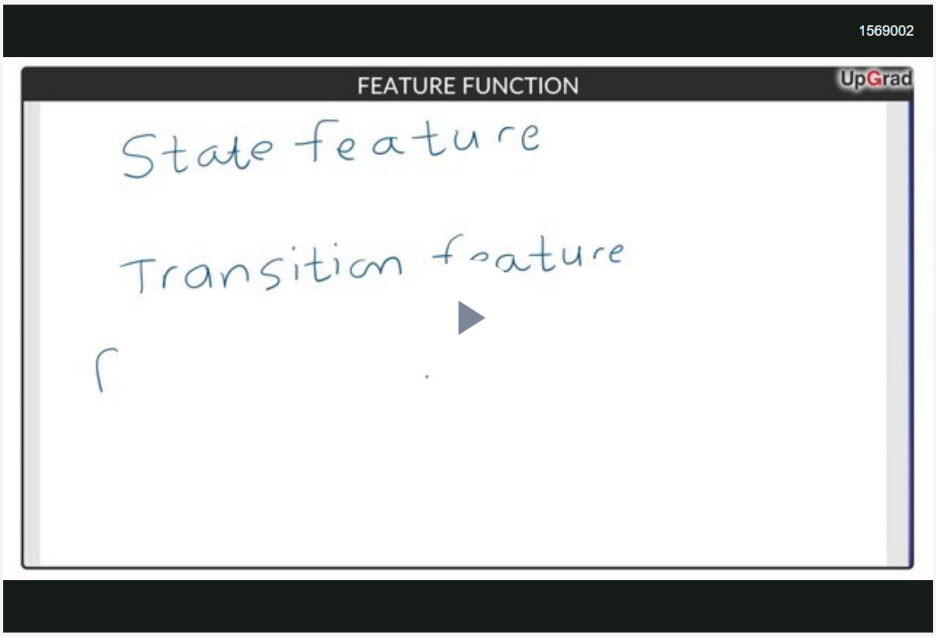
$$\textbf{sequence\_score}(y|x)$$

$$= \prod_{i=1}^n(exp(w.f(y_i,x_i,y_{i-1},i))) = exp(\sum_1^n(w.f(y_i,x_i,y_{i-1},i))) = exp(w.f(x,y))$$

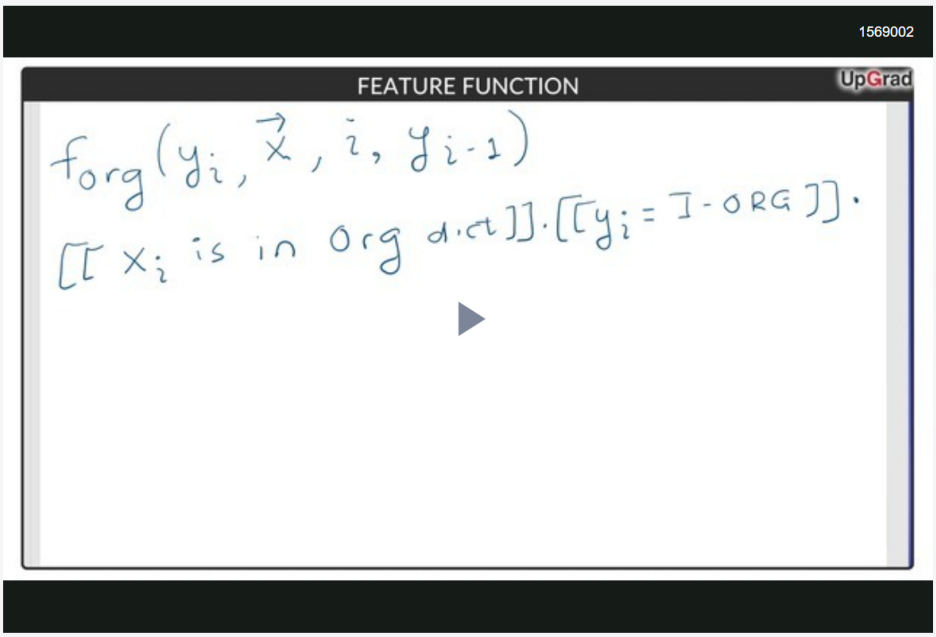
Then, $Z(x)$, if there are N possible sequences ($N = t^n$), the sum of scores of all possible N sequences is given by:

$$Z(x) = \sum_1^N(exp(w.f(x,y))$$

You'll also learn that there are two types of feature functions - **state features** and **transition features**.



You saw an example of a state feature. Let's now see an example of a transition feature which uses the previous label:



Question 1/2

Mandatory

Feature Functions- CRF

For transition probability, which of following parameters of feature function must be defined

y_{i-1} ✓ Correct

Feedback:
Label of the previous word in a sequence is important for defining transition probability

- i
- x_i
- None of the above

Your answer is Correct. Attempt 1 of 2 Continue >

Additional Reading

- In the next section, we will use concepts taught in logistic regression (specifically, maximum likelihood estimation). In case you are not familiar with those concepts, we highly recommend going through the optional module here (you will not be tested on any optional content).
- You can read more on the architecture of CRFs, how they are used for NER from the following document:

NER using CRFs Download

Report an error

PREVIOUS
Introduction to Conditional Random Fields (CRFs)

NEXT
Training a CRF model

CRF Model Architecture

In this segment, you'll study the CRF model architecture in detail. Let's start with CRFs' '**feature functions**'.

CRF Feature Functions

We had mentioned in the previous segment that CRFs use 'feature functions' rather than the input word sequence **x** itself. The idea is similar to how we had extracted features for building the naive Bayes and decision tree classifiers in a previous section. Some example 'word-features' (each word has these features) are:

- Word and POS tag based features: word_is_city, word_is_digit, pos, previous_pos, etc.
- Label-based features: previous_label

Let's spend some time to understand CRF features in detail.

Consider a sequence of n tokens $x = (x_1, x_2, x_3, \ldots x_n)$ and the corresponding IOB label sequence $y = (y_1, y_2, y_3, \ldots y_n)$. The **task** is to predict the label sequence y using the word sequence x.

Similar to conventional classifiers, we define some feature functions in CRFs which we'll denote as f. A feature function takes the following **four inputs**:

1. The input sequence of words: x
2. The position i of a word in the sentence (whose features are to be extracted)
3. The label y_i of the current word (the target label)
4. The label y_{i-1} of the previous word

There are usually multiple feature functions, let's denote the j^{th} feature function as $f_j(x, i, y_i, y_{i-1})$.

We restrict the model to extract label-based features using only the previous label (and not using labels of the previous n words where $n > 1$). This assumption is similar to the first-order Markov assumption we had used in POS tagging (i.e. transition probabilities depend only on the previous POS tag). Also, the feature functions return a real-valued number, which is often 0 or 1 (i.e. binary output).

Let's take an example of a feature function f_1 which returns **1** if the word x_i is a city **and** the corresponding label y_i is 'I-location', else 0. This can be represented as:

$f_1(x, i, y_i, y_{i-1})$ =[[x_i is in city list name] & [y_i is I-location]]

The feature function returns 1 only if both the conditions are satisfied, i.e. when the word is a city name and is tagged as 'I-location' (e.g. Chicago/I-location).

Similarly, a second feature function can be defined using the previous label, for e.g.:
$f_2(x, i, y_i, y_{i-1})$ = [[y_{i-1} is B-location] & [y_i is I-location]]

You can define multiple such feature functions f_i, though all are not equally important. Every feature function f_i has a **weight** w_i associated with it, which represents the 'importance' of that feature function. This is almost exactly the same as logistic regression where coefficients of features represent their importance.

Training a CRF means to **compute the optimal weight vector** w which best represents the observed sequences y for the given word sequences x. In other words, we want to find the set of weights w which maximises $P(y|x, w)$. This is similar to training a logistic regression model to find the optimal coefficients by maximising $P(y|x, w)$ where w represents the vector of logistic regression model coefficients. You'll study CRF model training shortly, for now, let's focus on the model architecture itself.

Recall that in logistic regression, the conditional probability $P(y|x, w)$ is modelled using a sigmoid function.

In CRFs, we model the conditional probabilities $P(y|x, w)$ in a similar fashion. If there are k feature functions (and thus k weights), for each word i in the sequence x, we define a **scoring function** as follows:

$$score_i = exp(w_1.\, f_1 + w_2.\, f_2 \ldots + w_k.\, f_k) = exp(w.\, f(y_i, x_i, y_{i-1}, i))$$

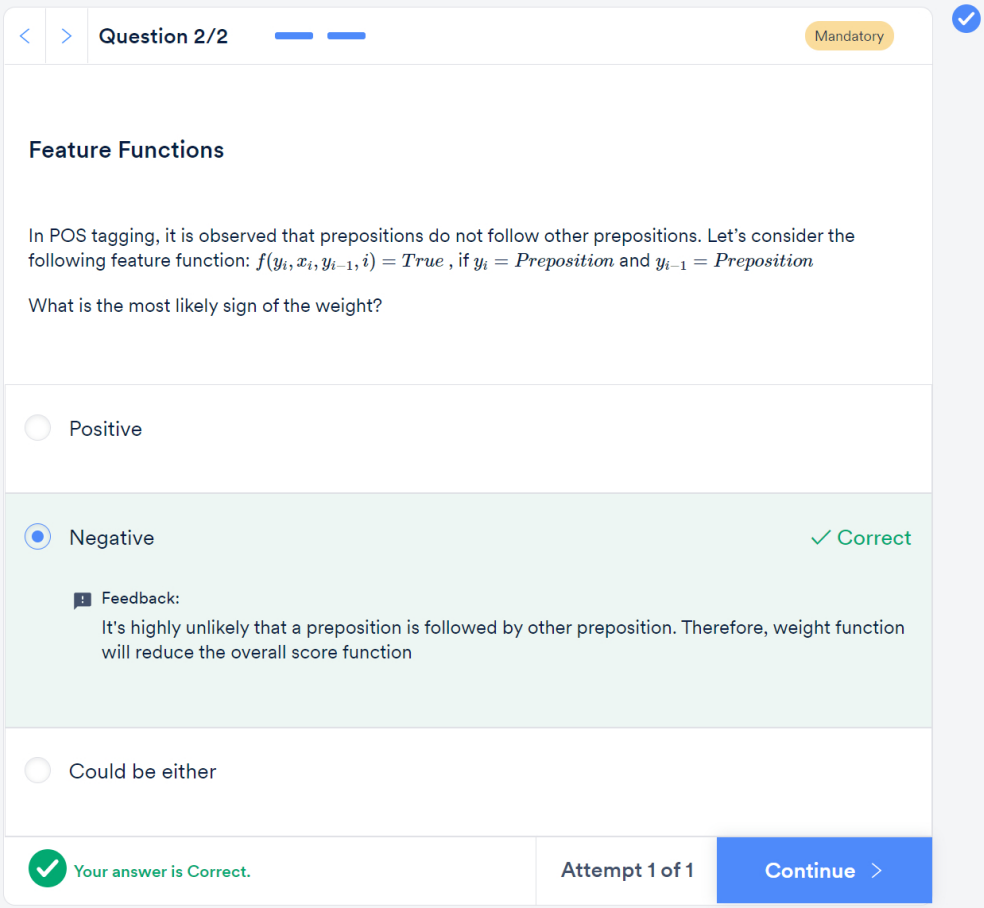
The intuition for **the score of the i^{th} word** is that it returns a high value if the correct tag is assigned to the word, else a low value. For e.g. say you want to tag the word 'Francisco' in the sentence 'The flight to San Francisco.' Consider that you have only two feature functions f_1 and f_2 defined above. Now, if you assign the (correct) label 'I-location' to 'Francisco', both f_1 and f_2 return a 1, and they return a 0 for any other tag such as 'O'. Also, $exp(x)$ being a monotonically increasing function, it returns a high value for high values of x. Thus, the score of a word is high when the correct tag is assigned to a word.

The score above is defined for each i^{th} word in the sequence x. If the sequences x and y are of length n, then the score of the sequence y is given by the **product of scores** of individual words in x. The sequence score will be highest when all the words have been assigned the correct labels:

$$\textbf{sequence\_score}(y|x) = \prod_{i=1}^{n}(exp(w.\, f(y_i, x_i, y_{i-1}, i)))$$

The product of exponents can be expressed as the exponential of the sums as follows:

$$\text{sequence\_score}(y|x) = \prod_{i=1}^{n}(exp(w.\, f(y_i, x_i, y_{i-1}, i))) = exp(\sum_{1}^{n}(w.\, f(y_i, x_i, y_{i-1}, i)))$$



Scores to Probabilities

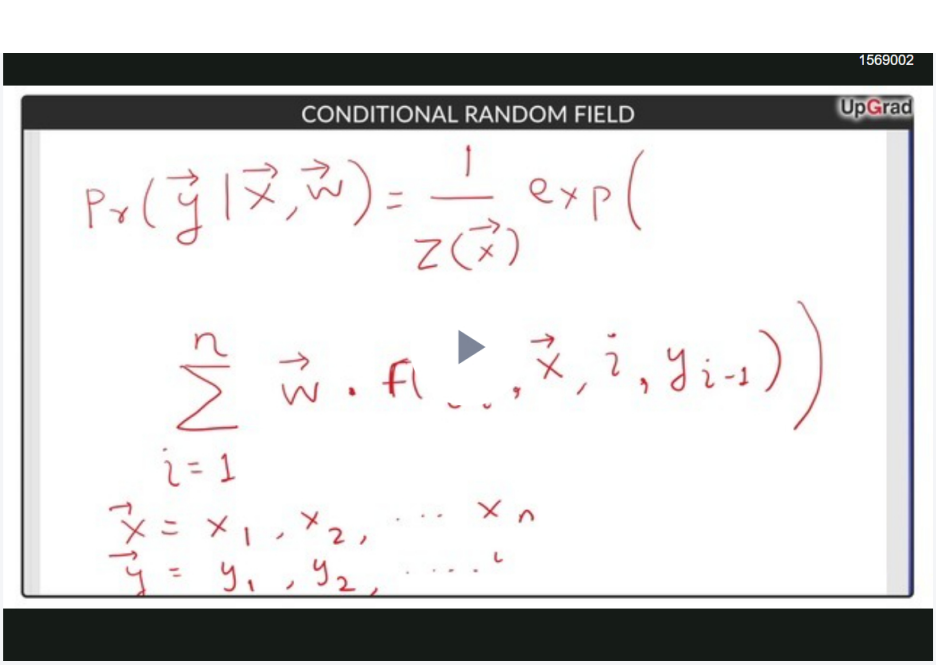
Using some math, we can now translate the scores of label sequences, which can be any real numbers, to probabilities. The probability $P(y|x, w)$, i.e. the probability of observing the sequence y given x for a certain w, is given by the score of the sequence y divided by the total score of all possible label sequences. If you have n words and t IOB labels, there are t^n possible IOB sequences y. The **inference task** is to find the optimal sequence y.

The denominator in the following expression, $Z(x)$, is a **normalising constant** which represents the sum of scores of all possible label sequences of x:

$$P(y|x, w) = exp(\sum_{1}^{n}(w.\, f(y_i, x_i, y_{i-1}, i)))/Z(x) = exp(w.\, f(x, y))/Z(x)$$

= score of sequence y assigned to x / sum of scores of all possible sequences

The term $exp(w.\, f(x, y))$ is a concise form of the expression $exp(\sum_{1}^{n}(w.\, f(y_i, x_i, y_{i-1}, i)))$ which represents the score of the sequence y assigned to x. Let's have Ashish explain this further.



To summarise, the probability of observing the label sequence y given the input sequence x is given by:

$$P(y|x,w) = exp(\sum_1^n(w.\,f(y_i,x_i,y_{i-1},i)))/Z(x) = exp(w.\,f(x,y))/Z(x)$$

= score of sequence y assigned to x / sum of scores of all possible sequences

For now, assume that you have somehow computed the optimal weights w by training the model - you'll study that in the next segment. For a sentence of n words and t IOB labels, there are t^n possible label sequences, and your task is to find the optimal sequence y_{opt} . You will study how to do this in a later segment.

Let's now understand the mathematical form of the normalising constant $Z(x)$ and some examples of feature functions. You have already seen that $Z(x)$ is the sum of scores of all possible label sequences. Since the probability of a tag sequence y is given by

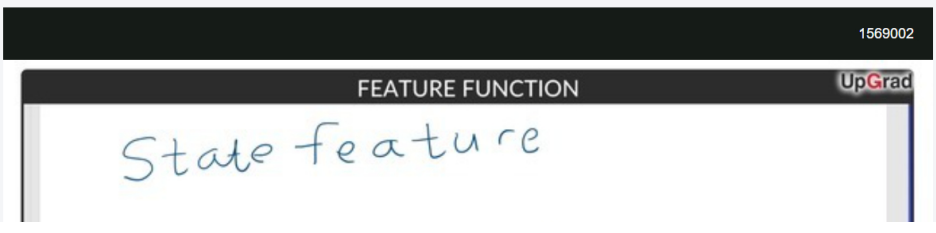
$$\textbf{sequence\_score}(y|x)$$

$$= \prod_{i=1}^n(exp(w.\,f(y_i,x_i,y_{i-1},i))) = exp(\sum_1^n(w.\,f(y_i,x_i,y_{i-1},i))) = exp(w.\,f(x,y))$$

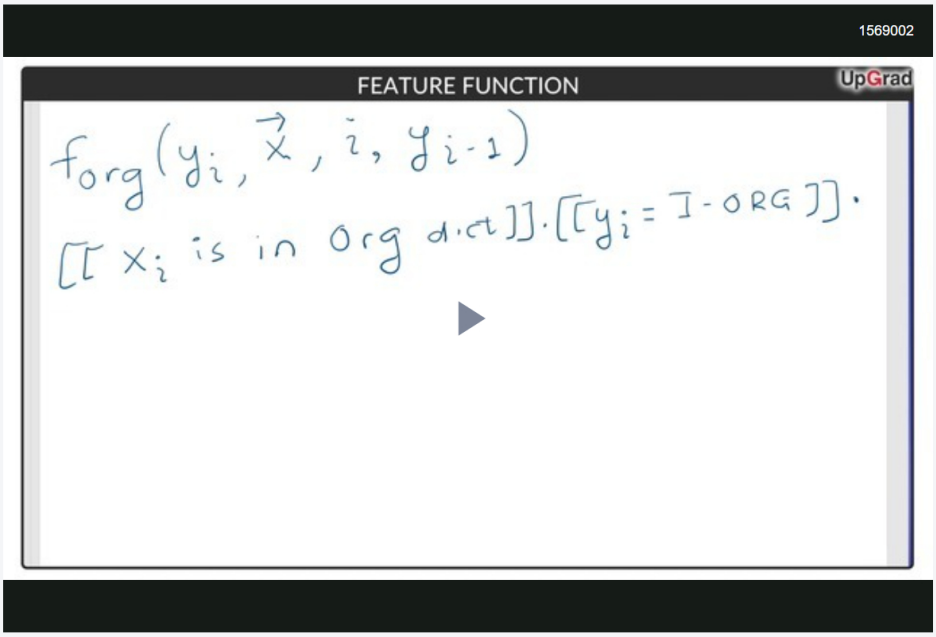
Then, $Z(x)$, if there are N possible sequences ($N = t^n$), the sum of scores of all possible N sequences is given by:

$$Z(x) = \sum_1^N(exp(w.\,f(x,y))$$

You'll also learn that there are two types of feature functions - **state features** and **transition features**.



You saw an example of a state feature. Let's now see an example of a transition feature which uses the previous label:



Question 2/2 — Mandatory

CRF

HMMs have two kinds of probabilities associated with it: Transition and Emission Probability. Select the option that correctly builds the analogy between HMMs and CRFs.

- Transition probability = State features, Emission probability = transition features
- Transition probability = transition features, Emission probability = state features — ✓ Correct

Feedback:
State features are dependent on word structures, morphologies, POS etc, similar to emission from word to tag

There is no similarity between these two models

Your answer is Correct. Attempt 1 of 1 Continue >

Additional Reading

- In the next section, we will use concepts taught in logistic regression (specifically, maximum likelihood estimation). In case you are not familiar with those concepts, we highly recommend going through the optional module here (you will not be tested on any optional content).
- You can read more on the architecture of CRFs, how they are used for NER from the following document:

NER using CRFs Download

Report an error

PREVIOUS
Introduction to Conditional Random Fields (CRFs)

NEXT
Training a CRF model

Training a CRF model

In the previous segment, we had mentioned that training a CRF model means to **compute the optimal set of weights** w which best represents the observed sequences y for the given word sequences x. In other words, we want to find the set of weights w which maximises the conditional probability $P(y|x,w)$ for all the observed sequences (x,y).

Recall that in logistic regression, we maximise the likelihood function:

$$L(w|x,y) = P(y|x,w)$$

where w represents the model coefficients. In other words, we compute the weights w such that the likelihood of observing the data points (x,y) is maximised for some set of weights w. We use exactly the same idea here - we compute the weights w such that likelihood $P(y|x,w)$ is maximised.

Note: To understand the difference between the terms 'likelihood' and 'probability', you can refer to the following brief explanation.

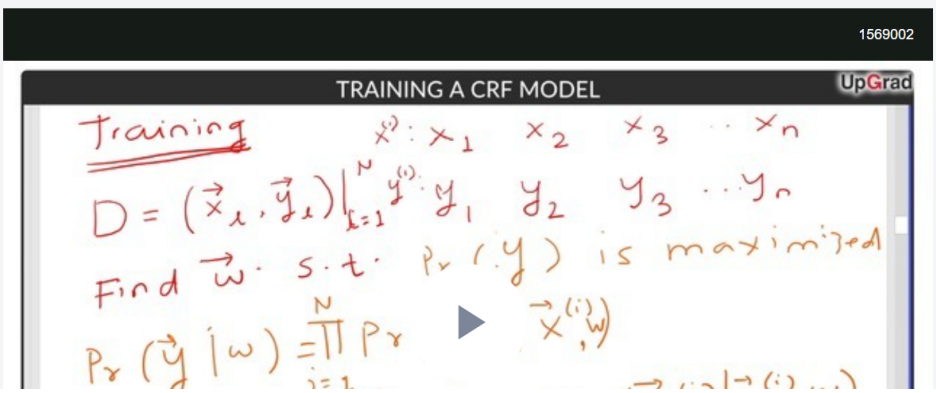
If there are N such sequences (i.e. N sentences along with their IOB labels), assuming that the sequences are independent, we want to maximize the product of likelihoods of all sequences, i.e. $\prod_1^N(P(y|x,w))$.

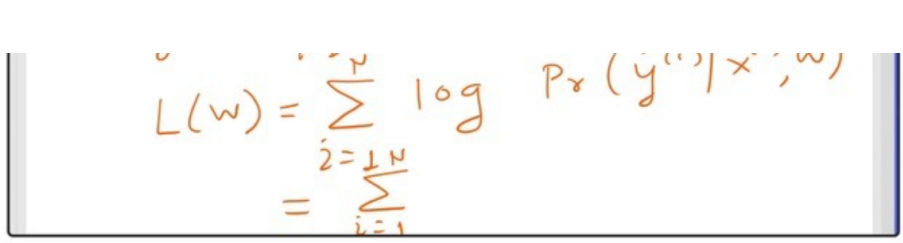
We have already established that the probability of observing a single sequence y assigned to x is:

$$P(y|x,w) = exp(\sum_1^n(w.f(y_i,x_i,y_{i-1},i)))/Z(x) = exp(w.f(x,y))/Z(x)$$

= score of sequence y assigned to x / sum of scores of all possible sequences

Now, if you have N such (x,y) sequences, the training task is to find weights w which maximise the probability of observing the N sequences (x,y). Let's have Ashish explain this in detail.





To summarise, the objective is to find weight vector w such that $P(y|x,w)$ is maximised. If there are N sequences, assuming that the N sequences are independent of each other, the likelihood function to be maximised is:

$$L(w|x,y) = P(Y|X,w) = \prod_1^N (P(y|x))$$

Since the likelihood function is exponential in nature, we take a log on both sides to simplify the computation (this makes sense since $log(x)$ is a monotonically increasing function, and thus, maximising x is equivalent to maximising $log(x)$):

$$L(w|x,y) = log(P(Y|X,w)) = \sum_1^N (log(P(y|x,w))$$

From the previous segment, $Pr(y|x,w)$ can be written as:

$$P(y|x;w) = exp(\sum_1^n (w.\, f(y_i, x_i, y_{i-1}, i))) / Z(x) = exp(w.\, f(x,y)/Z(x)$$

So, the final equation becomes:

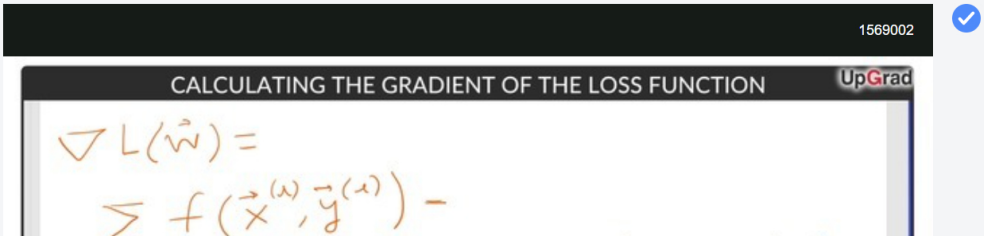
$$L(w|x,y) = \sum_1^N (log(exp(w.\, f(x,y)/Z(x)))$$
$$= \sum_1^N (log(exp(w.\, f(x,y)) - log(Z(x))) = \sum_1^N (w.\, f(x,y) - log(Z(x)))$$

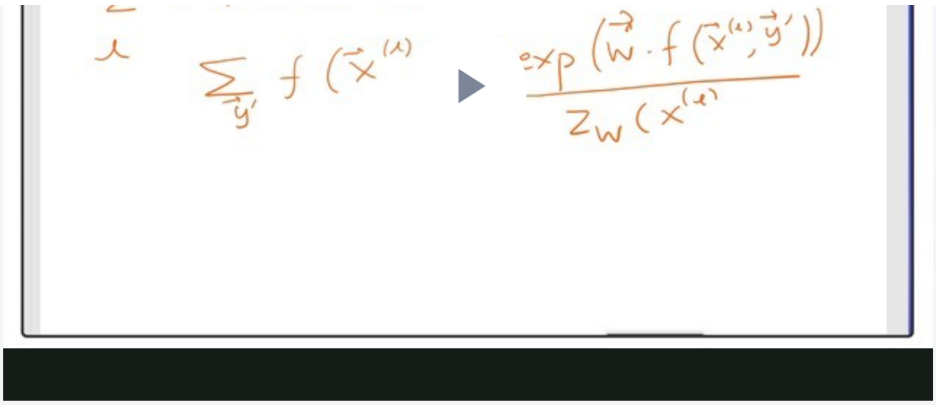
To prevent overfitting, we use the regularization term:
$L(w) = \sum_1^N [(w.\, f) - log(Z)]$ - regularisation_term

Refer to the advanced regression module for a refresher on regularisation. Briefly, **L1** and **L2** regularisation, also called Lasso and Ridge respectively, use the L-1 norm $(sum(|w|)$ and L-2 norm $(sum(w^2))$ respectively as the penalty terms.

Thus, we now have the objective function to be maximised.

Now, you have already studied **gradient descent** as an optimisation technique. In the upcoming lecture, you'll study how to maximise the log-likelihood equation using gradient descent:





The final equation after taking the **gradient of the log-likelihood function** is:

$$L(w) = \sum_1^N (f(x,y)) - E_{p_v(y|x,w)}(f(x,y^{'})) - 2w/C$$

where $E_{p_v(y|x,w)}(f(x,y^{'})) = \sum_1^N f(x,y^{'}) * exp(w.f(x,y^{'}))/z_w(x)$

So, we start with a random initial value of the weights w , and in each iteration, we adjust w to move in the direction of increasing cost. This direction is basically given by the gradient of the log-likelihood function.

Once you have maximised the likelihood function to find the optimal set of weights, you can use them to infer the labels for a given word sequence. In the next segment, you will learn how to do that.

Report an error

PREVIOUS
CRF Model Architecture

NEXT
Predicting using CRF

Predicting using CRF

In the previous segment, you have learnt that the optimal weights w can be computed by maximising the log-likelihood. Once the weights w are obtained, the next task is to assign the sequence y for any given word sequence x, i.e. assigning IOB tags to each word in the sentence x.

Question 1/3 — Mandatory

CRF

Once the weights w are obtained, the inference task is to assign the sequence y for any given word sequence x , i.e. assigning IOB tags to each word in the sentence x.

This is equivalent to:

- (●) Assigning y such that the score $exp(w.f(x,y))$ is maximum — ✓ Correct

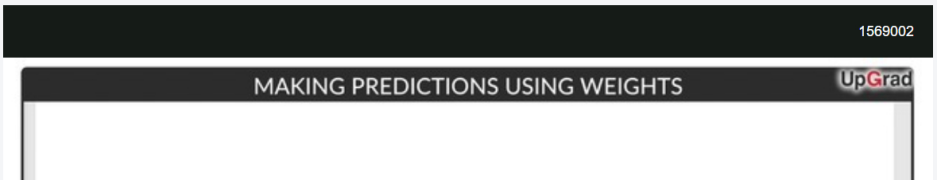
  Feedback:
  Label sequence y is assigned by maximising the scoring function

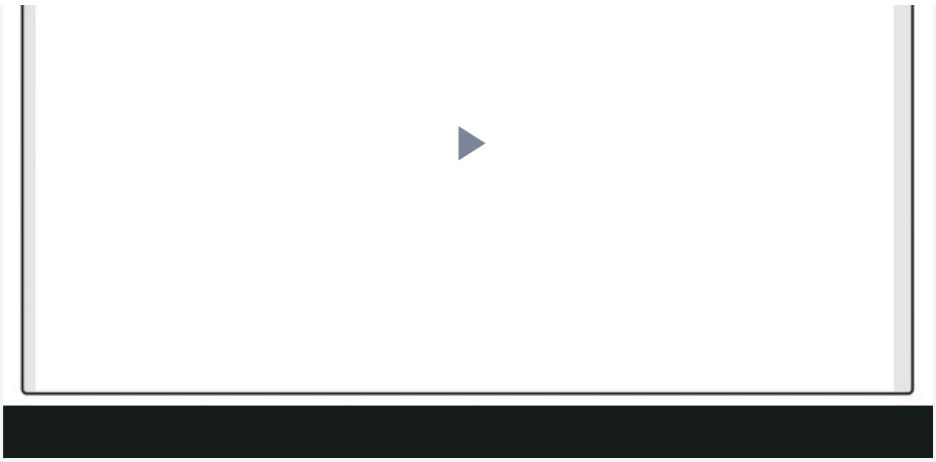
- ( ) Assigning y such that the score $exp(w.f(x,y))$ is minimum
- ( ) Assigning y such that the weights w are maximum
- ( ) None of the above

Your answer is Correct. Attempt 1 of 2 Continue >

Ashish will now explain how the assignment of labels can be done efficiently.

Correction: Ashish has defined the third feature function f_{dict} as $f_{dict} = [x_i \; in \; city_{dict}]$, though, it should be $f_{dict} = [x_i \; in \; city_{dict}]$ & $[y_i = C]$





Correction at 0:40: $y^* = arg\max_w(w.f(x,y))$

Thus, the inference task to assign the label sequence y^* to x which maximises the score of the sequence, i.e.

$$y^* = argmax(w.f(x,y))$$

The naive way to get y^* is by calculating $w.f(x,y)$ for every possible label sequence y, and then choose the label sequence that has maximum ($w.f(x,y)$) value. However, there are an exponential number of possible labels (t^n for a tag set of size t and a sentence of length n), and this task is computationally heavy.

Question 1/1 — Mandatory

CRFs vs HMMs

Which option describes the difference between HMM and CRFs? (Select all that apply)

- HMMs are used for sequential modelling, whereas CRFs are used when there are independent observations (random order)
- HMM puts a constraint on the current label to be dependent only on the current word and previous label. While CRFs can use more features like all the words in a sentence, etc. — ✓ Correct

  Feedback:
  HMM does put a constraint that word is dependent only on the current word and the previous tag
- HMMs are discriminative models and CRFs are generative models
- All of the above

Your answer is Correct. Attempt 1 of 2 Continue >

Previously, Ashish had defined three feature functions with weights as follows. Also, for simplicity, assume that there are only possible labels 'O' for outside an entity and 'C' for city:

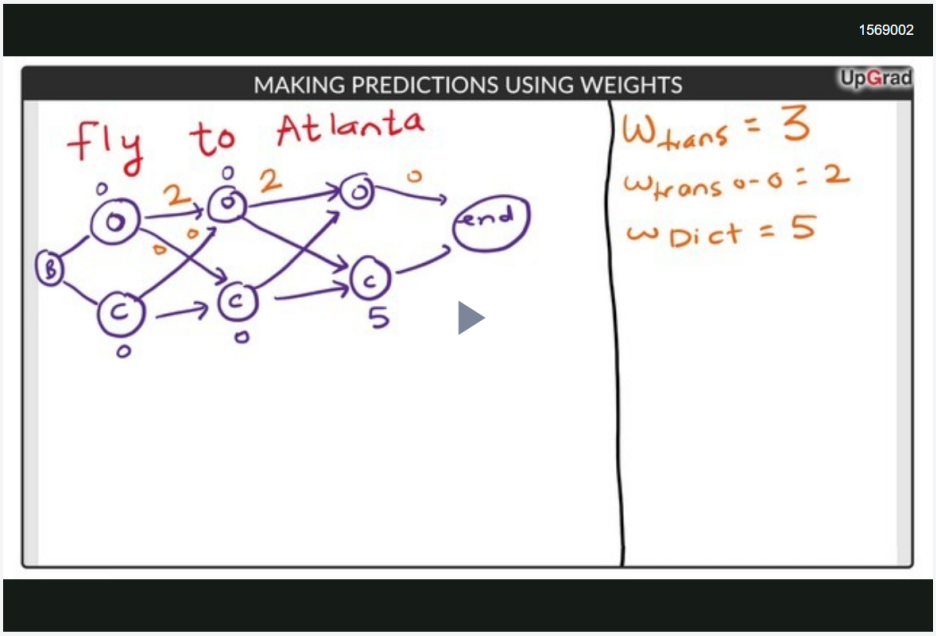
- $w_{trans} = 3; F_{trans} : y_{i-1} = O\,,\ x_i\ \textit{is in}\ city_{dict}\,,$
- $w_{trans} = 2;\ \&\ y_i = O$
- $w_{dict} = 5; f_{dict} = x_i\ \textit{is in}\ city_{dict}\ \&\ y_i = C$

The first feature will return 1 if the label is C, the previous label is O (i.e. it transitions from O to C), and if the word is in the city dictionary. Since the weight of the first feature function is 3, it 'fires' 3 if these conditions are satisfied.

Similarly, you can interpret the other feature functions. The second function returns 2 if two 'O's appear consecutively, while the third function returns 5 if the word is in the city dictionary.

Ashish will demonstrate this using an example.

[Slight correction: At 3:20, Ashish has written the weight 2 on the O-C edge, though, it should be on the O-O edge.]



You'll learn more about dynamic programming in the next lecture. Let's first solve some questions to understand how CRFs assign IOB tags to input sequences using weights and feature functions. The solution to these questions is provided below as a downloadable file.

CRF trellis

Consider the following phrase: 'Cheapest/JJS flight/NN from/IN Mumbai/NNP' for which the following three labels are to be tagged using CRF.

The three possible labels are as follows: flight_cost FC, city_name C, O

The model has the following feature functions and weights which are learnt from some training corpus:

1. F_{dict} : $y_i = C$, x_i is in the dictionary, w_{dict} =4
2. F_{trans} : y_i =city_name, $y_{i-1} = O$, $w = 6$
3. F_{state1}: y_i = flight_cost, $POS(x_i) = JJS$, $w = 3$,
4. F_{trans2} : $y_i = O, y_{i-1} = O, w = 1$

Consider that 'Mumbai' is in the city dictionary. Also, assume that the transitions from 'begin state' to any state and from any state to the 'end state' have 0 weights.

Calculate the score of the label sequence: O-O-O-city_name

- (●) 12 — ✓ Correct

  Feedback:
  O->O =1, O->O =1, O->city_name =6, city_name =4

  1+1+4+6=12

- ( ) 11
- ( ) 8
- ( ) 10

Your answer is Correct. | Attempt 1 of 2 | Continue >

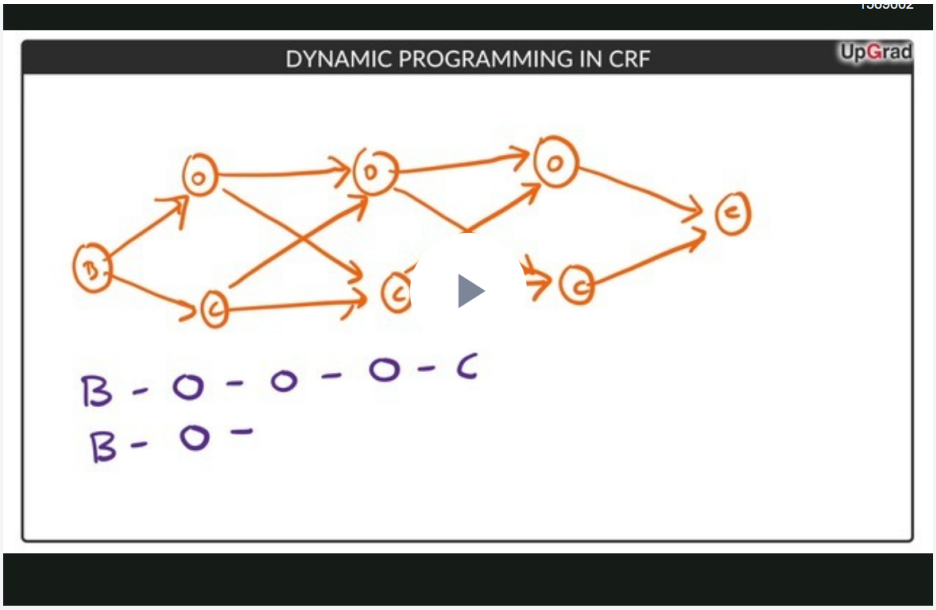
Solutions to the Questions

The following attached file will elucidate the **solution of the questions.**

Solution- CRF trellis — Download

In the next lecture, Ashish will give you an intuition of how to identify problems that require a dynamic programming algorithm. Dynamic programming applied to sequence prediction problems such as HMMs, CRFs etc. are commonly known as the **Viterbi algorithm.**

[Slight correction: At 0:55, Ashish has written the last state as C, but it should be E.]



In the next segment, Ashish will demonstrate how to build and evaluate CRFs in python on the airlines reservation dataset.

Report an error

PREVIOUS
Training a CRF model

NEXT
Python Implementation of CRF

Predicting using CRF

In the previous segment, you have learnt that the optimal weights w can be computed by maximising the log-likelihood. Once the weights w are obtained, the next task is to assign the sequence y for any given word sequence x, i.e. assigning IOB tags to each word in the sentence x.

Question 2/3 Mandatory

CRF

Once the weights w are obtained, the inference task is to assign the sequence y for any given word sequence x, i.e. assigning IOB tags to each word in the sentence x. This task is analogous to what you had done in HMMs while assigning POS tags to a sentence. Let's say there are n words in x and t possible tags/IOB labels.

The main problem with this task is:

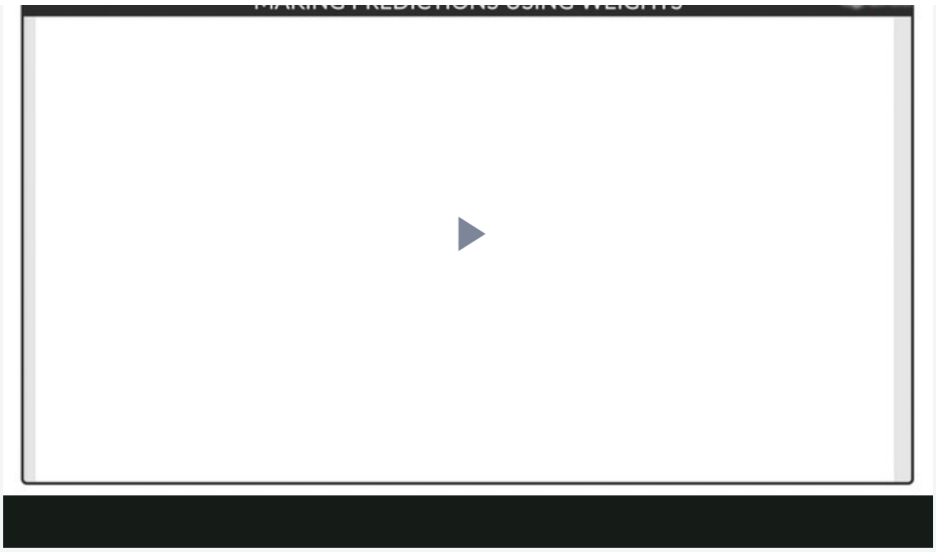
- Computational, since the total number of possible sequences, is n^t
- Computational, since the total number of possible sequences, is t^n ✓ Correct

  Feedback:
  For each word in a sequence of length n, t computations need to be done
- The weights are not known and we need to infer them
- None of the above

Your answer is Correct. Attempt 1 of 2 Continue >

Ashish will now explain how the assignment of labels can be done efficiently.

Correction: Ashish has defined the third feature function f_{dict} as $f_{dict} = [x_i \ in \ city_{dict}]$, though, it should be $f_{dict} = [x_i \ in \ city_{dict}]$ & $[y_i = C]$

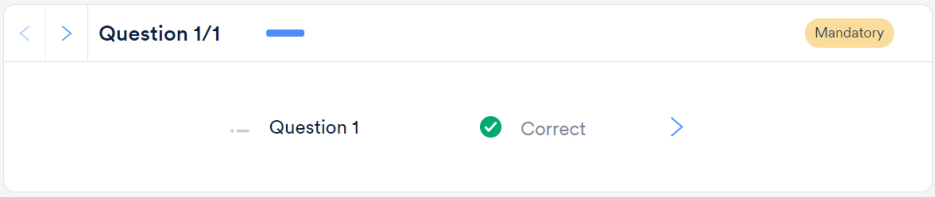


Correction at 0:40: $y^* = arg\max_w(w.f(x,y))$

Thus, the inference task to assign the label sequence y^* to x which maximises the score of the sequence, i.e.

$$y^* = argmax(w.f(x,y))$$

The naive way to get y^* is by calculating $w.f(x,y)$ for every possible label sequence y, and then choose the label sequence that has maximum ($w.f(x,y)$) value. However, there are an exponential number of possible labels (t^n for a tag set of size t and a sentence of length n), and this task is computationally heavy.



Previously, Ashish had defined three feature functions with weights as follows. Also, for simplicity, assume that there are only possible labels 'O' for outside an entity and 'C' for city:

- $w_{trans} = 3; F_{trans} : y_{i-1} = O\ ,\ x_i\ is\ in\ city_{dict}\ ,$
- $w_{trans} = 2;\ \&\ y_i = O$
- $w_{dict} = 5; f_{dict} = x_i\ is\ in\ city_{dict}\ \&\ y_i = C$

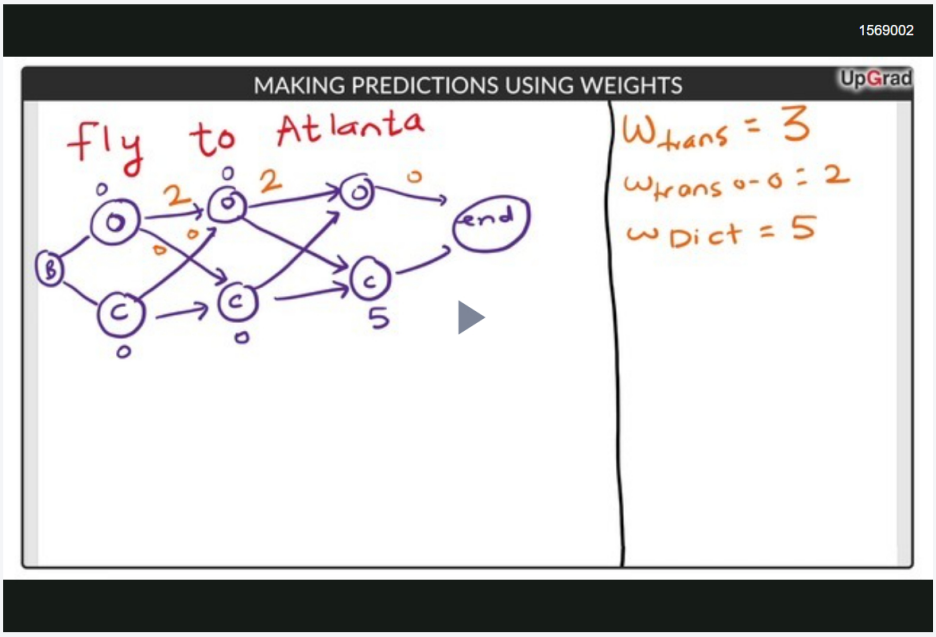
The first feature will return 1 if the label is C, the previous label is O (i.e. it transitions from O to C), and if the word is in the city dictionary. Since the weight of the first feature function is 3, it 'fires' 3 if these conditions are satisfied.

Similarly, you can interpret the other feature functions. The second function returns 2 if two 'O's appear

consecutively, while the third function returns 5 if the word is in the city dictionary.

Ashish will demonstrate this using an example.

[Slight correction: At 3:20, Ashish has written the weight 2 on the O-C edge, though, it should be on the O-O edge.]



You'll learn more about dynamic programming in the next lecture. Let's first solve some questions to understand how CRFs assign IOB tags to input sequences using weights and feature functions. The solution to these questions is provided below as a downloadable file.

Question 2/3 Mandatory

CRF trellis

Consider the following phrase: 'Cheapest/JJS flight/NN from/IN Mumbai/NNP' for which three following three labels need to be tagged using CRF.

There are three possible labels: flight_cost FC, city_name C, O

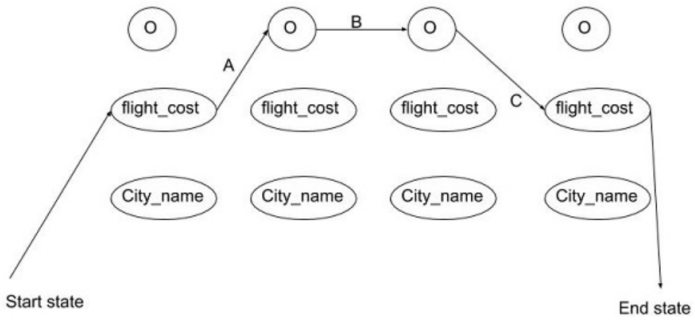
CRF is designed using below feature functions and weights

1. $F_{dict}: y_i = C$, x_i is in the dictionary, w_{dict} =4
2. $F_{trans}: y_i =$city_name, $y_{i-1} = O$, $w = 6$
3. F_{state1}: y_i = flight_cost, $POS(x_i) =' JJS'$, $w = 3$,
4. $F_{trans2}: y_i = O, y_{i-1} = O, w = 1$

Consider that 'Mumbai' is in the city dictionary. Also, assume that the transitions from 'begin state' to any state and from any state to the 'end state' have 0 weights.

Refer to the path shown in trellis below: start-state -> A->B ->C -> end-state

Cheapest flight from Mumbai



Calculate score of the path start-state -> A->B ->C -> end-state

4 ✓ Correct

Feedback:
flight_cost = 3, flight_cost -> O = 0, O -> O =1, O -> flight_cost = 0

3+1 =4

5

13

12

Your answer is Correct. Attempt 4 of 4 Continue >

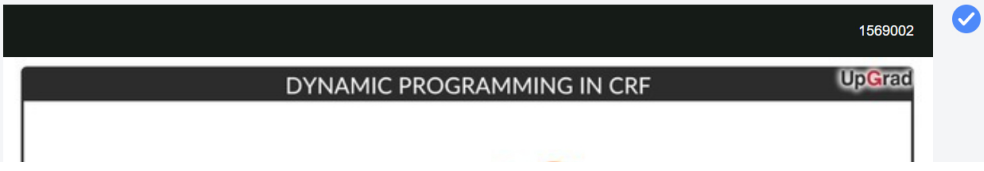
Solutions to the Questions

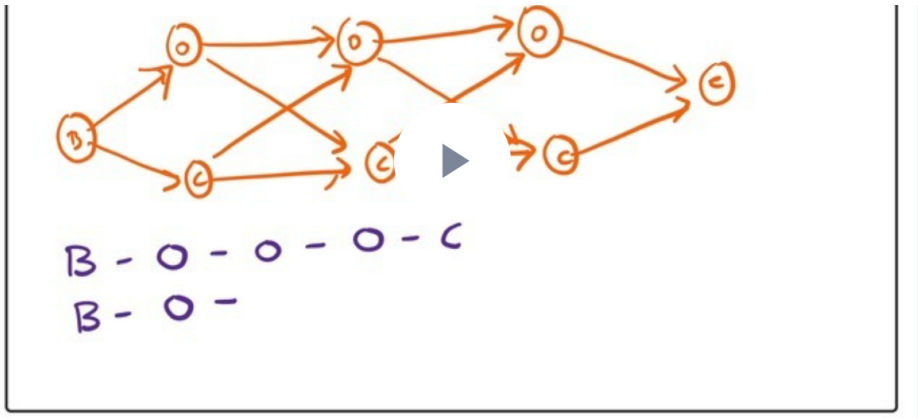
The following attached file will elucidate the **solution of the questions.**

Solution- CRF trellis Download

In the next lecture, Ashish will give you an intuition of how to identify problems that require a dynamic programming algorithm. Dynamic programming applied to sequence prediction problems such as HMMs, CRFs etc. are commonly known as the **Viterbi algorithm.**

[Slight correction: At 0:55, Ashish has written the last state as C, but it should be E.]





In the next segment, Ashish will demonstrate how to build and evaluate CRFs in python on the airlines reservation dataset.

Report an error

PREVIOUS
Training a CRF model

NEXT
Python Implementation of CRF

Predicting using CRF

In the previous segment, you have learnt that the optimal weights w can be computed by maximising the log-likelihood. Once the weights w are obtained, the next task is to assign the sequence y for any given word sequence x, i.e. assigning IOB tags to each word in the sentence x.

Question 3/3 — Mandatory

CRF

Which algorithm should you consider using to find the optimal label sequence in a computationally efficient manner?

Word Count 9 | Word Limit 1 - 10

Which algorithm should you consider using to find the

Suggested Answer
Viterbi Algorithm

Your answer is Correct. | Attempt 1 of 1 | Continue

Ashish will now explain how the assignment of labels can be done efficiently.

Correction: Ashish has defined the third feature function f_{dict} as $f_{dict} = [x_i \ in \ city_{dict}]$, though, it should be $f_{dict} = [x_i \ in \ city_{dict}]$ & $[y_i = C]$

1569002

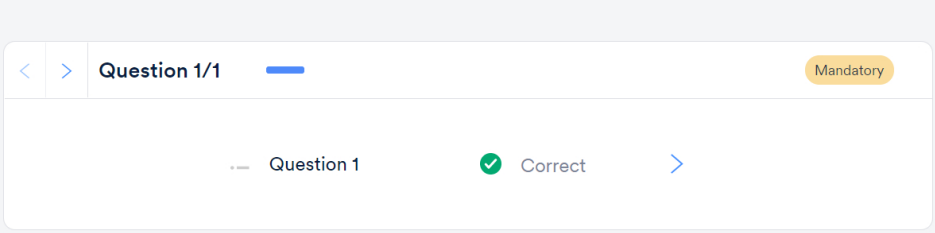
MAKING PREDICTIONS USING WEIGHTS — UpGrad

Correction at 0:40: $y^* = arg\max_w(w.\ f(x, y))$

Thus, the inference task to assign the label sequence y^* to x which maximises the score of the sequence, i.e.

$$y^* = argmax(w.\ f(x, y))$$

The naive way to get y^* is by calculating $w.\ f(x, y)$ for every possible label sequence y, and then choose the label sequence that has maximum ($w.\ f(x, y)$) value. However, there are an exponential number of possible labels (t^n for a tag set of size t and a sentence of length n), and this task is computationally heavy.



Previously, Ashish had defined three feature functions with weights as follows. Also, for simplicity, assume that there are only possible labels 'O' for outside an entity and 'C' for city:

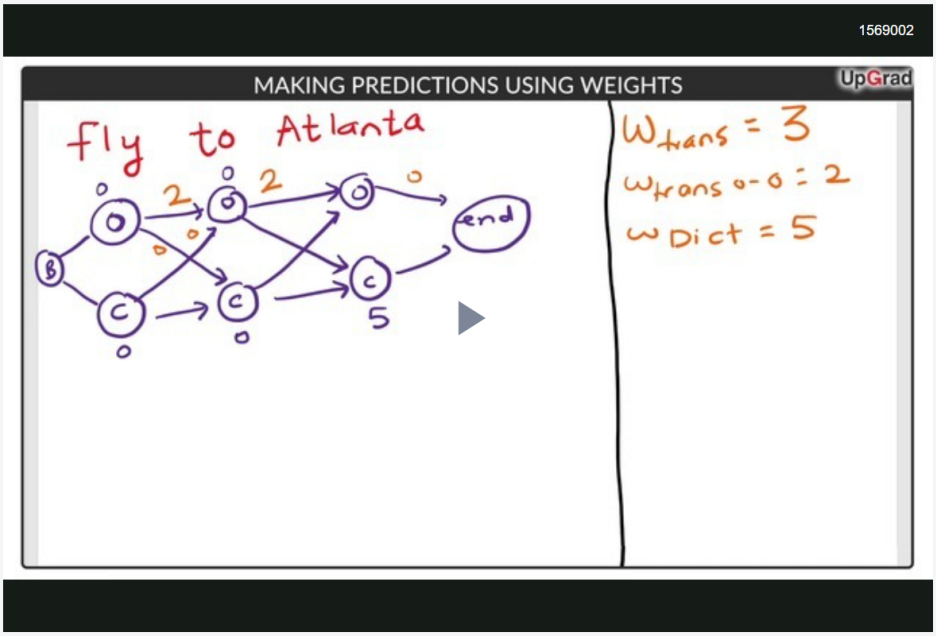
- $w_{trans} = 3; F_{trans} : y_{i-1} = O\ ,\ x_i\ is\ in\ city_{dict}\ ,$
- $w_{trans} = 2;\ \&\ y_i = O$
- $w_{dict} = 5; f_{dict} = x_i\ is\ in\ city_{dict}\ \&\ y_i = C$

The first feature will return 1 if the label is C, the previous label is O (i.e. it transitions from O to C), and if the word is in the city dictionary. Since the weight of the first feature function is 3, it 'fires' 3 if these conditions are satisfied.

Similarly, you can interpret the other feature functions. The second function returns 2 if two 'O's appear consecutively, while the third function returns 5 if the word is in the city dictionary.

Ashish will demonstrate this using an example.

[Slight correction: At 3:20, Ashish has written the weight 2 on the O-C edge, though, it should be on the O-O edge.]



You'll learn more about dynamic programming in the next lecture. Let's first solve some questions to understand how CRFs assign IOB tags to input sequences using weights and feature functions. The solution to these questions is provided below as a downloadable file.

Question 3/3 — Mandatory

CRF trellis

Consider the following phrase: 'Cheapest/JJS flight/NN from/IN Mumbai/NNP' for which three following three labels need to be tagged using CRF.

There are three possible labels: flight_cost FC, city_name C, O

CRF is designed using below feature functions and weights

1. F_{dict} : $y_i = C$, x_i is in the dictionary, w_{dict} =4
2. F_{trans} : y_i =city_name, $y_{i-1} = O$, $w = 6$
3. F_{state1}: y_i = flight_cost, $POS(x_i) =' JJS'$, $w = 3$
4. F_{trans2} : $y_i = O, y_{i-1} = O, w = 1$

Consider that 'Mumbai' is in the city dictionary. Also, assume that the transitions from 'begin state' to any state and from any state to the 'end state' have 0 weights. Find the most probable path.

- Start-state -> O -> O-> O -> city_name -> end-state
- Start-state -> flight_cost -> O-> O -> city_name -> end-state ✓ Correct

Feedback:
flight_cost=3, O-> O =1, O->city_name =6, city_name= 4

3+1+6+4= 14

Start-state -> O -> O-> O -> flight_cost -> end-state

Start-state -> flight_cost -> O-> O -> O -> end-state

Your answer is Correct. Attempt 2 of 2 Continue >

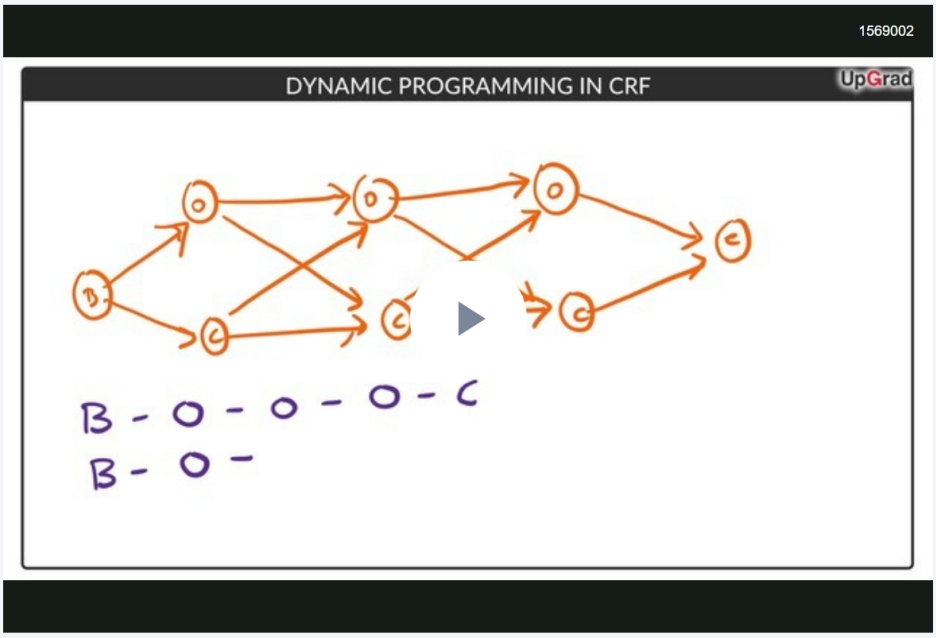
Solutions to the Questions

The following attached file will elucidate the **solution of the questions.**

Solution- CRF trellis Download

In the next lecture, Ashish will give you an intuition of how to identify problems that require a dynamic programming algorithm. Dynamic programming applied to sequence prediction problems such as HMMs, CRFs etc. are commonly known as the **Viterbi algorithm.**

[Slight correction: At 0:55, Ashish has written the last state as C, but it should be E.]



In the next segment, Ashish will demonstrate how to build and evaluate CRFs in python on the airlines reservation dataset.

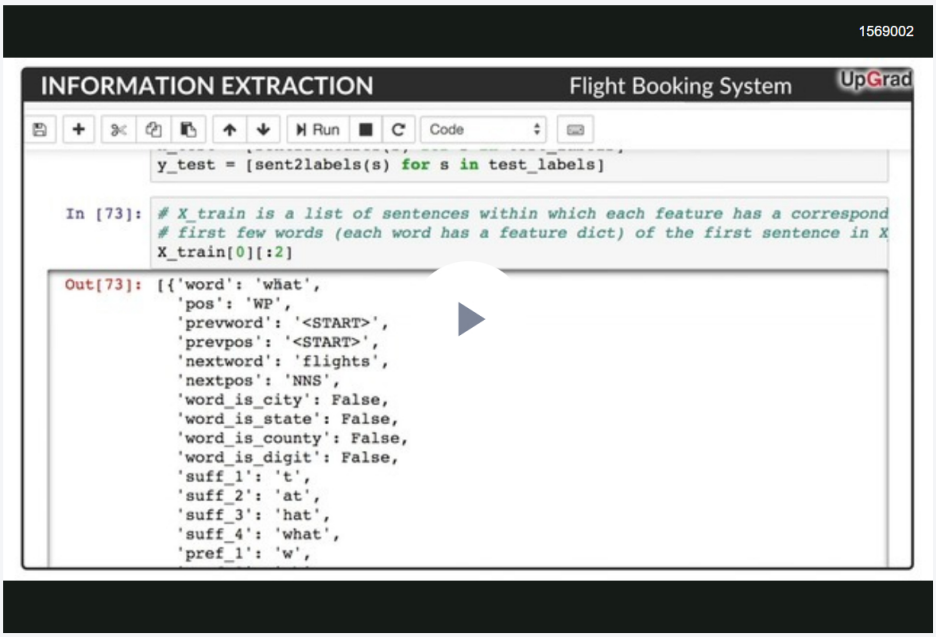
Report an error

PREVIOUS
Training a CRF model

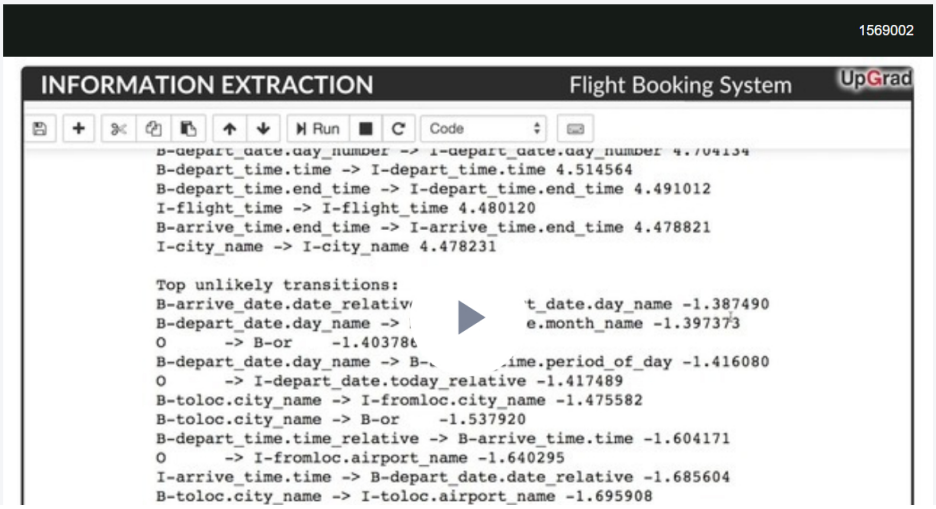
NEXT
Python Implementation of CRF

Python Implementation of CRF

Let's now study the Python implementation of CRF on the ATIS data. We will build a CRF model by defining some features f. Also, we will create some new features compared to what we had used previously, such as the suffix and prefix of the word (i.e. the first and the last few letters in the word; this is to just demonstrate that one can create such features based on word morphologies).



Let's now understand what the CRF classifier has learnt. First, we will save the trained CRF model in a pickle file, and then use it to make predictions on the test set.



```
B-arrive_time.time -> I-depart_time.time -1.743275
O      -> I-city_name -1.802009
O      -> I-toloc.city_name -1.802546
```

This brings us to the end of CRFs and the session on Information Extraction.

Building a Flight Booking Application - Optional

The code at the bottom part of the notebook (under the heading 'Building an Application') contains some extra code that is not covered in the lectures. That section is totally optional, though we recommend you to go through the code and experiment with it (only if you have extra time).

The idea of 'building an application' is that once you are able to parse the queries and extract entities from it, you can use the entities to query a database or an external API which can take the entities extracted by your model as input and return a list of flights (i.e. schedules) which you can suggest to the user. Such a system can be used to build, say, a flight-booking chatbot.

In this case, we have used the flightstats API to get flight schedules. In the code, you will observe that this task involves a number of non-trivial data processing/cleaning steps, such as:

- Resolving inconsistencies in data format:
  - The API needs the date of flight in a certain format, such as 'dd/mm/yyyy', so you need to convert entities such as 'the thirtieth of August' to '30/08/2018' etc. (this is a fairly non-trivial problem)
  - There are multiple airports in some cities (New York has three), and the API call returns a list of flights to/from all airports. You may want to filter the list to contain only the main airports before showing results to the user. Also, the API needs airport codes such as 'JFK' for the New York airport, so you need to map the city name to the corresponding airport code.
- Preprocessing:
  - User queries will be in the form of strings, not a list of tuples like we have in the training set. You need to word-tokenize the query, assign POS tags, and convert it to a suitable format to feed to the model

This exercise is intended to give you some exposure to the tasks that one needs to perform to build an ML-based product, though in many cases these are done by engineering teams, not data scientists.

PREVIOUS
Predicting using CRF

FINISH SESSION
Python Implementation of CRF