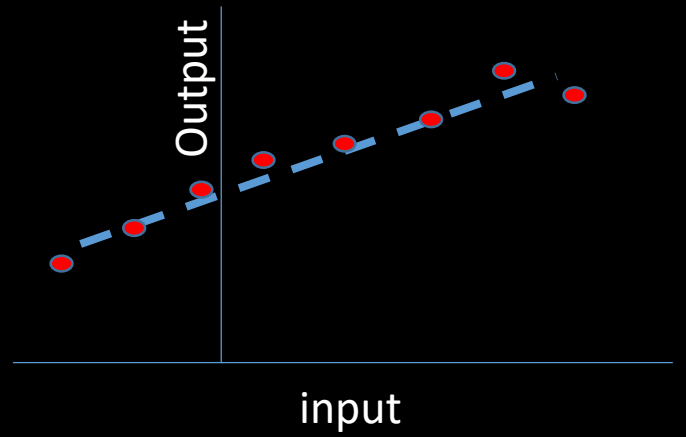
Linear Models

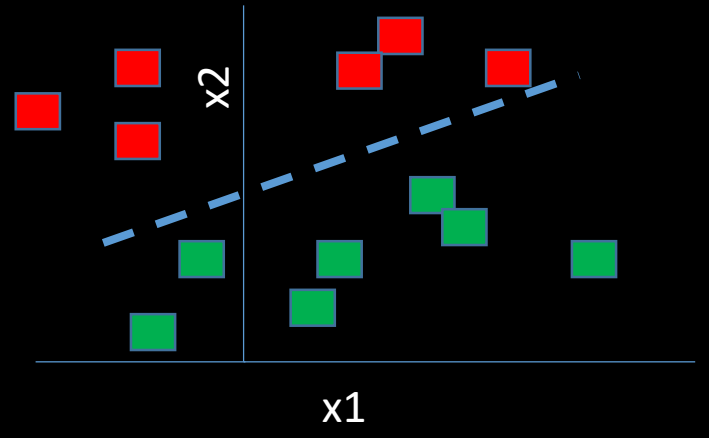
Palacode Narayana Iyer Anantharaman

9 July 2016

Linear Models

- Linear models constitute a space of **hypothesis** that assume the output of a system is a linear function of the input
- **Regression** problems are those with continuous variables as output
  - The term regression sometimes is confusing as some of us may think of regression as in “regression testing”, which is a different concept. In statistics and in machine learning, regression problems are simply those that produce real valued outputs.
- **Classification** problems are those that label the given input in to one or more labels from a finite label set
- **Linear Regression** is a machine learning technique that predicts an output assuming a linear relation between the output and input
- **Logistic Regression** is a classification technique assuming a linear decision boundary
  - Though a sigmoid function is used in Logistic Regression, it is still considered a linear model, as the decision surface is linear





Some Example applications of Linear Regression

- Time series predictions
  - TV advertising: Given the TRP rating data across a number of TV programs, can we determine the serial to advertise?
  - You are the director of a mega serial and you have the viewership information from past weeks. Can you predict the future trends and do some intervention as needed?

- Evaluating trends and making forecasts
  - Suppose your company offers infrastructure as a service over cloud and you are required to plan periodic upgrades to the hardware. Given the customer acquisition rate, usage patterns, etc. can you predict the capacity upgrade?

- Assessing credit limits
  - If you are a bank and have received a home loan application, can you determine the optimal loan amount and a custom EMI scheme that maximizes mutual benefit?

- Predict the onset of a disease
  - Given the historical data on patients with diabetes, predict the probability of a new patient getting this disease

Example: Banking Sector (Ref ToI, 12th July 2016)

Your Rs 10-lakh personal loan disbursed within 10 seconds

Mayur Shetty| TNN | Jul 12, 2016, 01.16 AM IST


A- A+

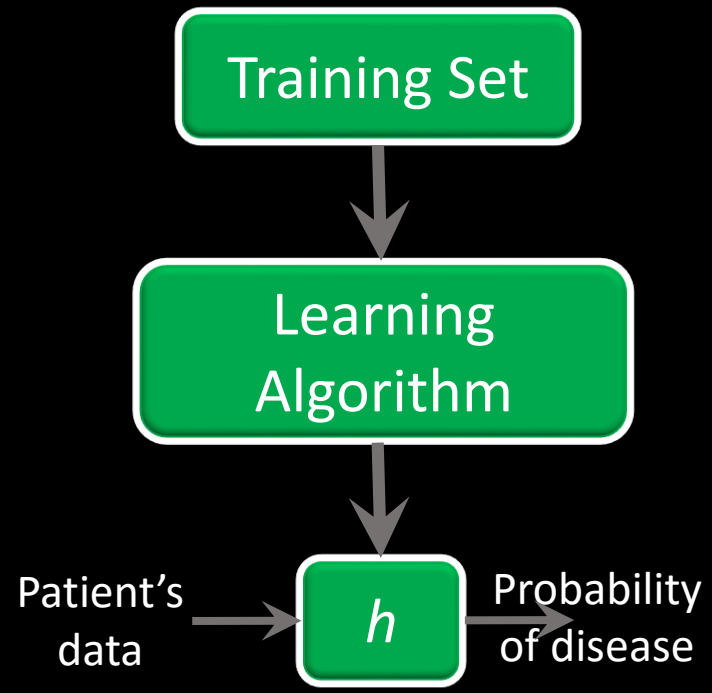
MUMBAI: Slowly, but surely, retail lending is being taken over by bots. And they are doing a good job of it. Banks are now moving beyond automation of small-ticket personal loans to parts of mortgages.

The early mover in using algorithms for lending is HDFC Bank, which started out with its 10-second loan. Earlier, obtaining a personal loan required the borrower to approach a customer service executive. This can now be done by accountholders online or even from an ATM. Despite the "unsecured" tag, the quality of personal loan portfolio is overall sound and defaults are very low. "The instant loan scheme

Citi's 'Instant Personal Loan', which is entirely algorithm-driven, accounts for 20% of its monthly personal loan bookings. The algo also determines the mortgage

Linear Regression Hypothesis

- A supervised machine learning hypothesis is a model that attempts to behave as the true but unknown target function: $f: X \rightarrow Y$
- Given the training data (inputs, targets), the model learns to emulate f by maximizing the probability of training data
- Linear regression assumes f to be a linear function as given by:
$$h(\theta) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n$$
- The values of θ are to be determined by the training procedure such that the hypothesis $h(\theta)$ approximates $f(x)$



Estimating Model Parameters

- At a high level, we need to:
  - Start with some initial model (that is some random values of theta)
  - Estimate the output
  - Determine the cost of misclassification (in the case of classification problems) or inaccurate estimation (in the case of regression problems)
  - Using the training data, iteratively modify the model parameters such that the cost is minimized
  - Output the model when the cost is minimum (that is when the iteration converges)

- Once the model is estimated it can be used for predicting/estimating new inputs (test data)

Cost Function for Linear Regression

- The cost function is often dictated by the requirements of the application. Though several widely used cost functions (e.g. mean squared error) exist, one may come up with his/her own if needed

- The most commonly used cost function for regression problems is the squared error, defines as:

$$J(\theta) = \frac{1}{2m} * \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$$

- Our goal is to minimize $J(\theta)$ with respect to θ

- $J(\theta)$ as a function of θ is a quadratic function. It is a convex function with a single minimum point and is differentiable.

Finding the minimum of J(θ)

- We need to find θ by minimizing: J(θ) = $\frac{1}{2m} * \sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$

- To determine the min we differentiate the cost function and set it to zero

- The above form of J(θ) is differentiable and can be solved analytically. For the ith training example and for the parameter θ_j, the derivative is:

$$\frac{\partial J}{\partial \theta_j} = \left(h_\theta(x^i) - y^i\right)\ x_j^i$$

- We use a numerical way of determining the differentials because:
  - Analytical solutions involve matrix multiplications and for a large dataset with a large number of training examples and features, there may be memory overflows
  - For some cost functions, closed form solutions are not available.

- Gradient Descent algorithm is widely used to compute parameter updates

Batch Gradient Descent Algorithm

$$J(\theta) = \frac{1}{2m} * \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$$

Repeat {

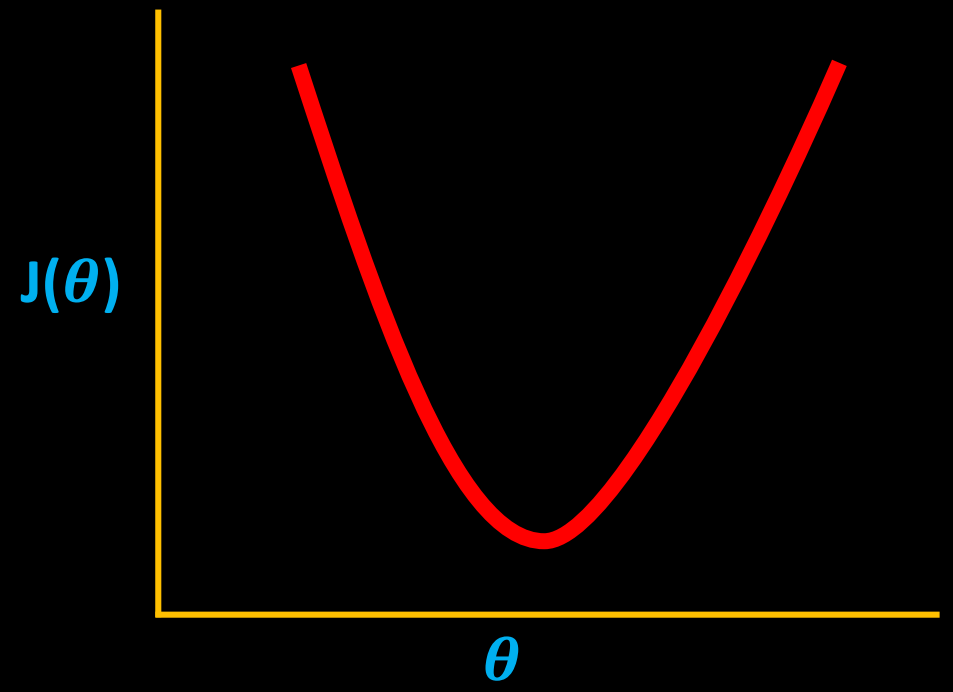
$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i$$

} for every j from 1 to n

α is the learning rate, which is a hyper parameter that is usually set to a small fraction

Learning Rate in Gradient descent

- It is necessary to choose the learning rate α to have an optimal value
- Small values of α result in more number of iterations for convergence while a large value of α may make the algorithm overshoot the minimum point and J(θ) may not decrease.
- To choose α try 0.001, 0.003 and so on



Effect of initialization

- Does the parameter initialization have an impact on determining the right model that has minimum cost?

- What happens if we set all parameters to zero during initialization?

Minimizing Loss: Gradient Descent

- Batch gradient descent
  - Given m training examples, compute J(θ) as the total cost across all examples and use this to perform parameter updates.
  - Batch gradient descent uses ALL the examples in order to perform a single parameter update.

- Minibatch gradient descent
  - Segment the m examples in to batches of a batch size b, compute J(θ) for a batch, update the parameters and do this for all batches (m/b)
  - Mini batch gradient descent uses b examples for a single parameter update

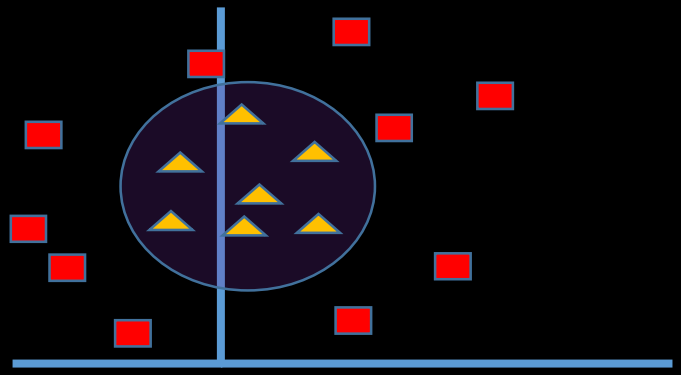
- Stochastic Gradient Descent
  - For each training example, compute the cost and update the parameters. This is equivalent to minibatch with a batch size b = 1

Applying Linear Regression to classification

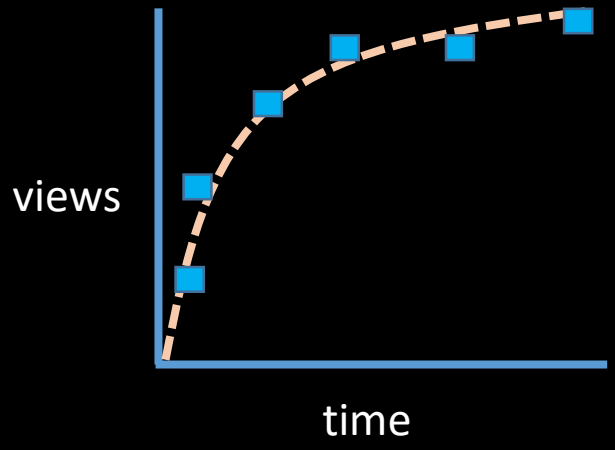
- As described, linear regression produces the output as a continuous variable. That is both h(x) as well as y are continuous variables.
- If we constrain y to be discrete, say -1 or 1, then the same linear regression model can be applied to classification problems also.
- However the decision surfaces are inflexible
- They can be used to generate the initial model parameters to another more specialized classification algorithm

Linear Regression for non linear features

- What if the underlying target function is a non linear function with respect to the features?



- We can still use linear models with a pre processing step
  - Suppose we have: $X = (x_1, x_2)$ and the output is a quadratic function of input.
  - Form new features: $(x_1, x_2, x_1^2, x_2^2, x_1 x_2)$ and corresponding theta parameters.
  - Train as before and get the new model
  - For prediction, perform same transformation and use the linear model as before



Case Study

- Problem
  - Suppose we have videos of a TV serial that get uploaded on the YouTube on a daily basis
  - Each day there will be a new episode and will be uploaded
  - Our goal is to predict the number of views of any given episode on the day 3 after it was uploaded.
  - Assume that we don’t have access to the special analytics that YouTube may be providing to their customers

- Solution
  - Model the growth rate of the number of views with respect to time. This can be done by visualizing the data and arriving at a suitable model.
  - Perform the transformation of input vectors as needed
  - Train the system with the data available from recent episodes
  - Validate, test

Logistic Regression

- Used for addressing classification problems

- Examples:
  - Email Spam classification
  - Loan approvals
  - Text classification

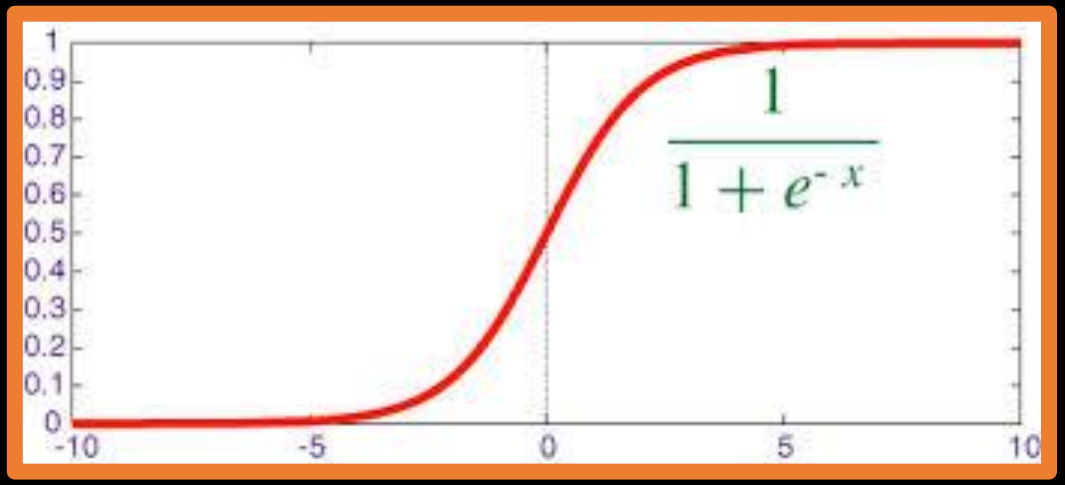
- Here the output is a discrete value. For binary classification, where the output is either 0 or 1, we have:

$$y \in \{0, 1\}$$

Logistic Regression: $0 \leq h_\theta(x) \leq 1$

Logistic Regression Hypothesis

- Logistic Regression is a linear regression model with a sigmoid at the output
  - Sigmoid is a squashing function that is differentiable with output bounds 0, 1



- $h_\theta(x)$ is the probability of output y = 1 on the given input x, parameterized by θ
  - Note: The logistic regression system outputs a valid probability

- **Question**: Suppose a function f(x) outputs a real number between 0 to 1 for any x. Can this be considered as generating a valid probability distribution?

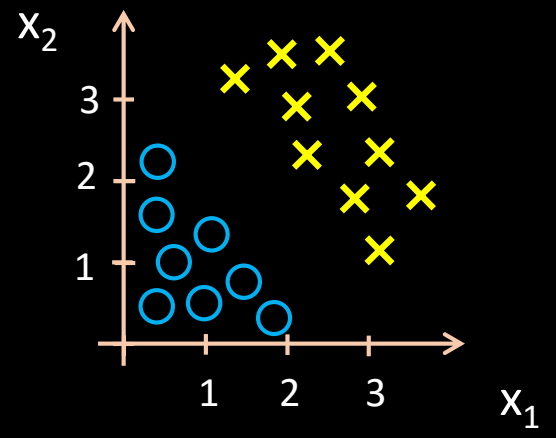
$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

Logistic Regression: Decision Boundary

- Predicts the output to be 1 if $h_\theta(x) >= 0.5$, predict 0 if $h_\theta(x) < 0.5$
- This implies: predict 1 if $\theta^T x >= 0$, predict 0 if $\theta^T x < 0$
- Example:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$



Let θ be the vector (-3, 1, 1). We have: $h_\theta(x) = -3x_0 + x_1 + x_2$

Predict y = 1 if $-3x_0 + x_1 + x_2 >= 0$, else predict y = 0

Logistic Regression Cost Function

The cost function for logistic regression is usually the **cross entropy error** function.

$$J(\theta) = -\frac{1}{m}\left[\sum_{1}^{m} y^i \log\ h_\theta(x^i) + (1 - y^i)\log(1 - (h_\theta(x^i))\right]$$

Let's take a brief digression to understand the cross entropy concept in the next 2 slides

Intuition behind entropy

- Consider a discrete random variable x. How much information we receive when we observe a specific value of x?
- The amount of information can be viewed as the degree of surpise on learning the value of x. That is a highly improbable information provides more information compared to a more likely event. When an event is certain to happen we receive zero information.
- Hence the information h(x) is a monotonic function of probability p(x)
- If there are 2 independent events, the information that we receive on both the events is the sum of information we gained from each of them separately. Hence: $h(x, y) = h(x) + h(y)$
- Two unrelated events will be statistically independent if: $p(x, y) = p(x)\ p(y)$
- From the above 2 relationships we deduce that h(x) should be related to log p(x). Specifically: $h(x) = - \log_2 p(x)$.

Entropy

- Suppose a sender transmits the value of a random variable to a receiver.
- The average amount of information transmitted is obtained by taking the expectation of h(x) with respect to the distribution p(x) and is given by:
- $H[x] = -\sum_x p(x)\, log_2\, p(x)$, $H[x]$ is called the Entropy of the random variable x
  - Consider a random variable that can assume one of 8 possible values, where each value is equally likely. Here, $H[x]$ is given by $-8 \times \frac{1}{8} \times log_2 \frac{1}{8}$ = 3 bits
  - Now consider the distribution to be: p(x) can be $\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}\}$. The entropy turns out to be 2 bits
- Thus, if a random variable has a uniform probability distribution, the associated entropy is maximum
- If p(x) and q(x) are two probability distributions occurring over the same set of events, we define the cross entropy to be:

$$H(p, q) = -\sum_x p(x)\, log_2\, q(x)$$

Cost Function - intuition

$$J(\theta) = -\frac{1}{m}\left[\sum_{1}^{m} y^i \log\ h_\theta(x^i) + (1 - y^i)\log(1 - (h_\theta(x^i))\right]$$

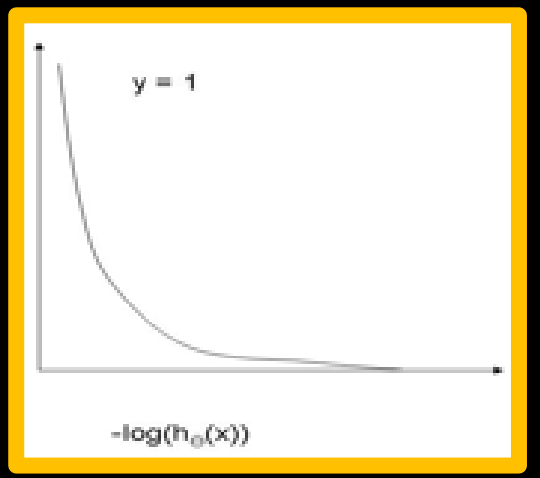
- The cost can be split in to two parts: cost incurred when true output y = 0 and the cost when y = 1

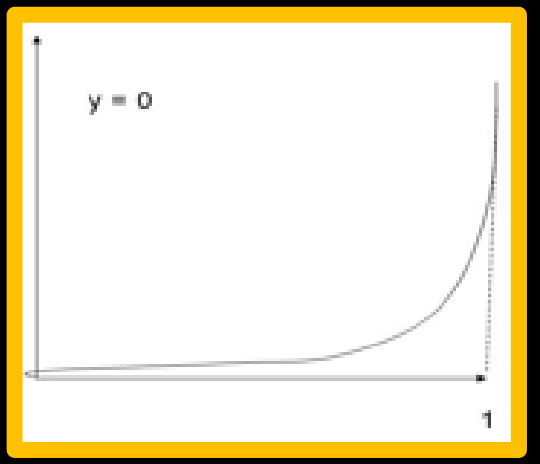
$$Cost(h_\theta(x), y) = -\log(h_\theta(x))\ when\ y = 1$$
$$Cost(h_\theta(x), y) = -\log(1 - h_\theta(x))\ when\ y = 0$$

- From the above it is easy to see:

  Cost = 0 when y = 1 and $h_\theta(x)$ = 1 but as $h_\theta(x) \to 0, Cost \to \infty$

  Similarly we can analyse the cost for the case where y = 0





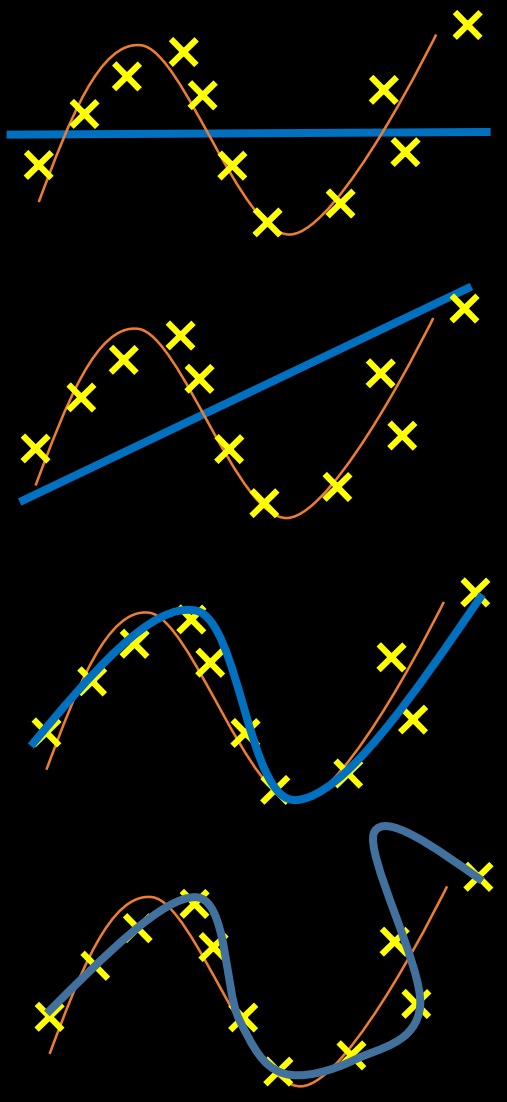
Gradient Descent for logistic regression

- Using the cross entropy error function with the logistic regression hypothesis function, we can show that the derivatives turn out to be of the same form as linear regression

- Hence the same gradient descent algorithm works for logistic regression with $h_\theta(x)$ replaced with the sigmoid

Overfitting and Underfitting

- Consider the adjoining figure where:
  - Let's say the unknown target function is close to a Sine wave and the training samples are shown as **x** marks
  - We fit different polynomials: constant, linear, cubic and higher order polynomials and the model is shown as a blue coloured curve



- We observe:
  - Fit with constant or linear is too simplistic
  - Fit with polynomials of high order, say, order = 9, are complex models and they try to pass through every sample in the training data.

Underfitting: High Bias Problem

- Fitting a simplistic (say, linear) model to a highly non linear underlying target function would result in high training error as well as test error.

- Though the training samples show a non linearity, we pretend as if everything is simple and try to fit the data with our biased view of the world! Due to this, we say this is a “bias” problem.
  - Suppose we need to do face recognition given an image, is it possible to use linear models and hope for great accuracy?

- This is termed underfitting or high bias

Overfitting: High Variance Problem

- If we use a complex model (say, a high order polynomial of degree > 5) we end up trying to account for every training sample.

- This is like fitting the data rather than emulating the underlying target process.
  - This kind of model can't see the wood for the trees!

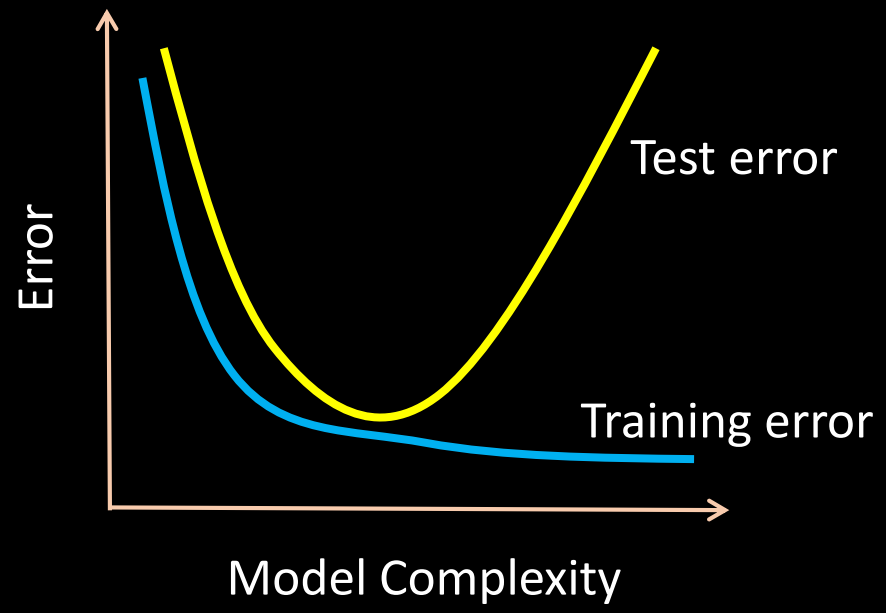
- This approach is error prone because:
  - This is very sensitive to noise and outliers
  - Fitting very tightly to the training data is memorizing and not generalizing!

Training and Test Errors

- When the model is too simple for a given application, both the training and test errors are high due to underfitting

- When the model is the right fit, the test error is at the minimum

- If we increase the model complexity still further, the training error continues to reduce but the test error shoots up



If a model fits the training data well, it doesn’t automatically mean that it would fit the test data well

How to avoid underfitting?

- Assuming we have adequate amount of training data, we could increase the model complexity
  - Use a higher order polynomial to fit the data
  - Use a more powerful model (Neural Nets, Deep Learning, etc)
  - Add more relevant features

How to avoid overfitting?

- When we use a complex model such as the one with higher order polynomials, we need to keep the weights (theta values) small.

- Without any constraints in choosing the parameters, complex models tend to have larger weights when they attempt to fit every training example.
  - This is similar to having too many knobs to turn and each knob set to whatever value without constraints in order to fit the data

- One way to minimize the impact of overfitting is to use regularization

Regularization

- Small values of parameters lead to simpler hypothesis and reduces overfitting

- We can keep the parameters under check by introducing a regularization term to the cost function as below:

$$\mathrm{J}(\theta) = \frac{1}{2m} * \left[\sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2 + \lambda \sum_i \theta_j^2 \right] \quad \lambda \, is \, the \, regularization \, coefficient$$

$$J(\theta) = -\frac{1}{m}\left[\sum_{1}^{m} y^i \log \; h_\theta\left(x^i\right) + \left(1 - y^i\right)\log\left(1 - \left(h_\theta\left(x^i\right)\right)\right)\right] + \frac{\lambda}{2m}\sum_{i} \theta_j^2$$

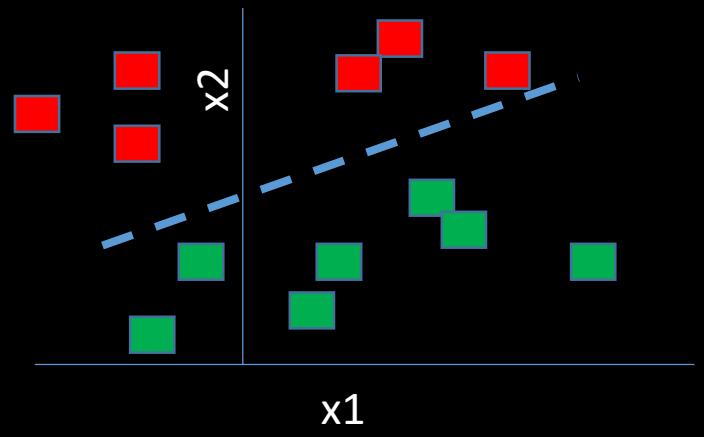
Multiclass Classification

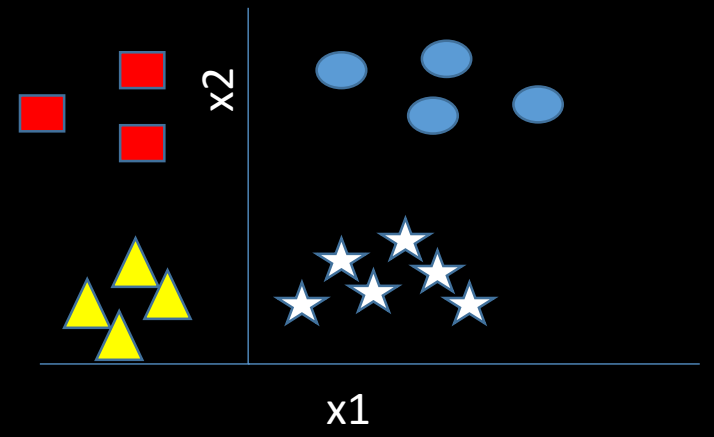
- The logistic regression we discussed is a binary classifier

- Many real world applications may require a multi class classification

- Examples
  - Text classification: Suppose we want to categorize a document in to (Technical, General, Sports, Movies, Business). If there is a news article that reads something like: “Sachin Tendulkar was awarded Bharat Ratna”, we might label this both as general news as well as an item pertaining to sports.

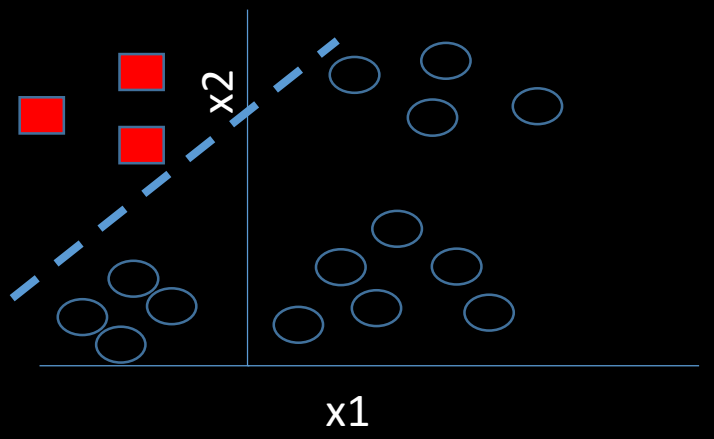
One vs All Logistic Classifier





- To perform multiclass classifier for K classes, build K independent binary logistic classifiers each with their parameter vectors.

- A binary classifier for label j is trained to predict for y = j and treat all other classes as not(y=j)



How does Naïve Bayes Relates to Logistic Regression?

- Some interesting questions to think about are:
  - Does logistic regression have equivalence to Naïve Bayes Classifier?
  - What kind of decision surface does the Naïve Bayes classify?

- With a few simple mathematical steps the equivalence between the two models can be demonstrated

Softmax Classifier

- A multiclass logistic regression allows labelling over K classes (instead of just binary) by running K independent regressors in parallel.

- Thus, the outputs $i \in K \; and \; j \in K \; where \; i \neq j$ are produced independently, except that they share the same input.

- On many situations, it may be more useful to generate a probability distribution over the output classes for a given input.

- Softmax classifier is a multinomial logistic regression classifier that addresses this requirement

Softmax Hypothesis

- Recall that the logistic is a linear regression with a sigmoid discriminator output
- Let the real valued output from the linear transformation $\theta^T x$ be a score s

$$s = f(X;\theta)\ where\ f\ is\ a\ linear\ function\ that\ transforms\ the\ input$$

- The softmax output is a probability distribution given by:

$$P(Y = k|X) = \frac{\exp(s_k)}{\sum_j \exp(s_j)}$$

- It is easy to show that P(Y|X) as above is a valid probability distribution.

Cost Function for Softmax

- The hypothesis expression of a softmax classifier is a probability distribution over the output classes

- Given the training data, we would like to maximize the likelihood or equivalently, the log likelihood
  - Log is a monotonic function of its input, maximizing log(x) is equivalent to maximizing x.
  - We use log due to mathematical convenience

- Maximizing log likelihood is same as minimizing the negative log likelihood (NLL)

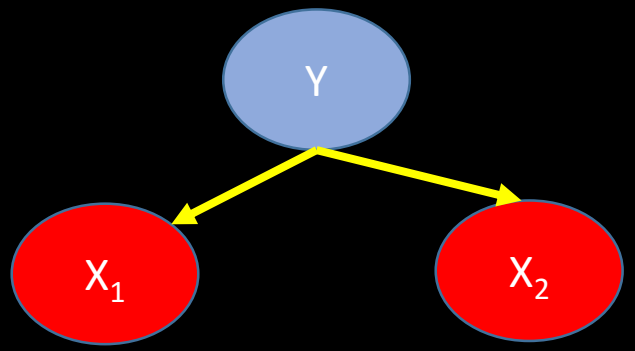
- Thus, our cost function is: $L_i = -\log(\ P\big(Y = y_i \big| X = X_j\big))$

$$L_i = -\log(\ P\big(Y = y_i \big| X = X_j\big)) = -\log(\frac{\exp(s_i)}{\sum_z \exp(s_z)})$$

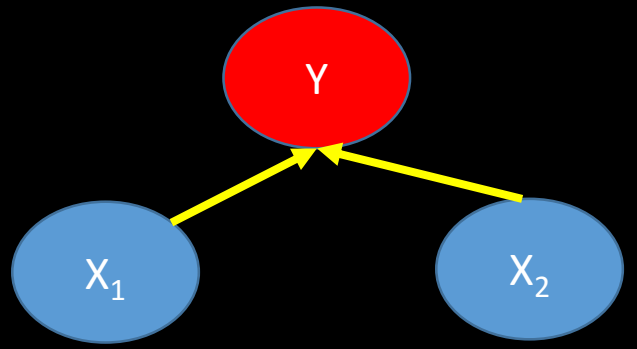
Code Walkthrough

Generative Versus Discriminative Models

- Key notion behind the generative models is that the abstract (or hidden or latent) state generates the observations. A given abstract state has a likelihood of generating the given observations.
  - Naïve Bayes Classifier is an example of generative model



- As opposed to the generative models, the discriminative models take the observations as given and try to figure out what hidden state might best explain the observations.
  - Logistic Regression is an example of discriminative model



Generative Versus Discriminative Models

- Generative models are characterized by joint distributions.
  - We ask the question: What is the probability of the observations X and the abstract label Y to co occur? That is: P(X, Y)

- Discriminative models are conditional models. They are characterized by the conditional probability distribution
  - Given the observations (features), what is the class label that is most likely?

- As we model the joint distribution in generative models, we need both the prior P(Y) as well as the likelihood P(X|Y) in order to determine the posterior P(Y|X)

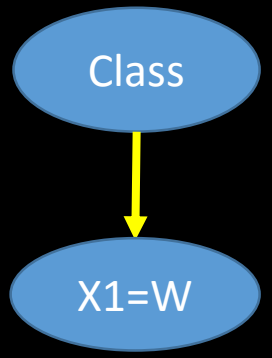
- Discriminative model do not need to model hidden states P(Y) as they directly learn P(Y|X) from the observations or features

Overcounting the evidence

- Suppose we have a vocabulary V with words: V = {“Sachin”, “Tendulkar”, “Wimbledon”} and two classes: C = {“Cricket”, “Tennis”}

Wimbledon Wimbledon	Wimbledon	Wimbledon Wimbledon	Wimbledon Virat Kohli
Wimbledon Virat Kohli	Wimbledon	Virat Kohli	Virat Kohli

- There are 4 documents on Cricket (Brown background) and 4 on Tennis (Blue background)



NB Factors:

P(C) = P(T) = 0.5

P(W|C) = 1/4

P(W|T) = 3/4

Joint Probs

P(T, W) = P(T) P(W|T) = 1/2 * 3/4 = 3/8

P(C, W) = P(C) P(W|C) = 1/2 * 1/4 = 1/8

Predictions

P(T|W) = ?

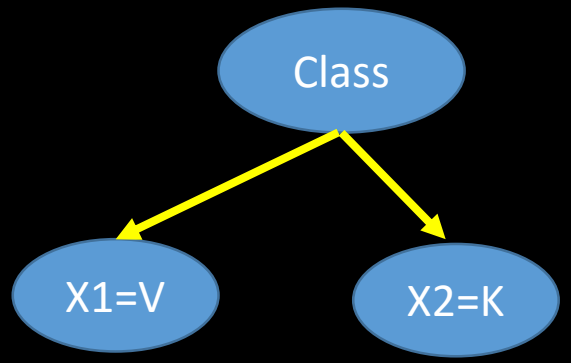
P(C|W) = ?

Overcounting the evidence

- Suppose we have a vocabulary V with words: V = {“Sachin”, “Tendulkar”, “Wimbledon”} and two classes: C = {“Cricket”, “Tennis”}

Wimbledon Wimbledon	Wimbledon	Wimbledon Wimbledon	Wimbledon Virat Kohli
Wimbledon Virat Kohli	Wimbledon	Virat Kohli	Virat Kohli

- There are 4 documents on Cricket (Brown background) and 4 on Tennis (Blue background)



NB Factors:

P(C) = P(T) = 0.5

P(V|C) = P(K|C) = 3/8

P(V|T) = P(K|T) = 1/8

Joint Probs

P(T, V, K) = P(T) P(V, K|T) = 1/2 * 1/8 * 1/8

P(C, V, K) = P(C) P(V, K|C) = 1/2 * 3/8 * 3/8

Predictions

P(T|V, K) = ?

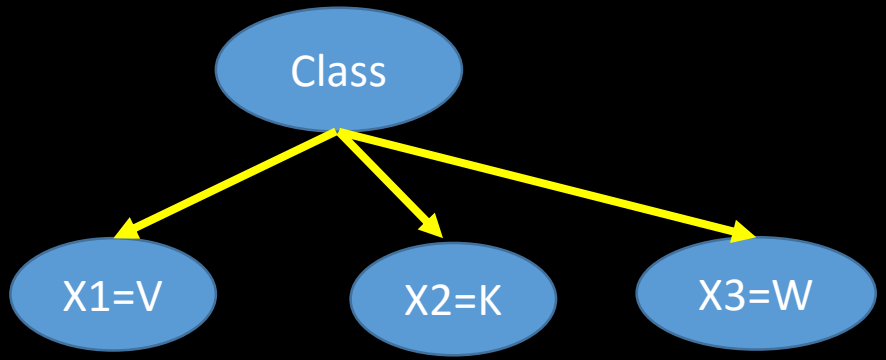
P(C|V, K) = ?

Overcounting the evidence

- Suppose we have a vocabulary V with words: V = {“Sachin”, “Tendulkar”, “Wimbledon”} and two classes: C = {“Cricket”, “Tennis”}

- There are 4 documents on Cricket (Brown background) and 4 on Tennis (Blue background)

Wimbledon Wimbledon	Wimbledon	Wimbledon Wimbledon	Wimbledon Virat Kohli
Wimbledon Virat Kohli	Wimbledon	Virat Kohli	Virat Kohli

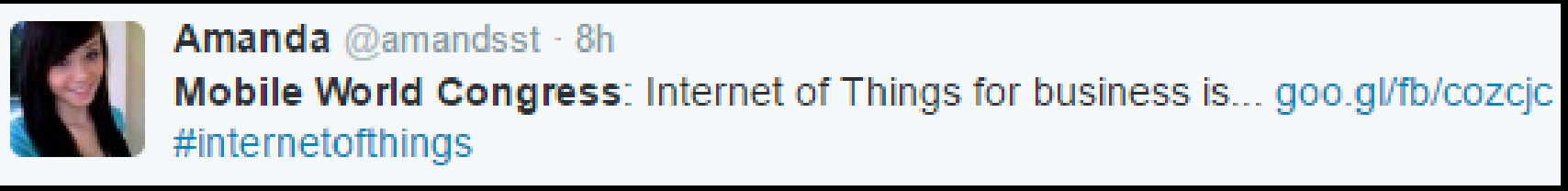


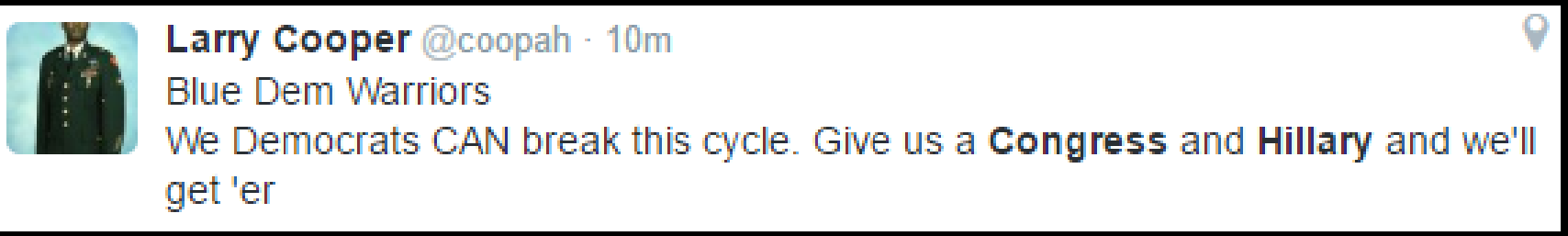
Predictions

P(T|V, K, W) = ?

P(C|V, K, W) = ?

Back to the earlier case study…





- How do we apply logistic or softmax regression to the twitter search problem we discussed?
- For a domain specific problem like this (where the labels are fixed and given), we can take advantage of some prior knowledge, instead of depending only on observed data
  - Example of prior knowledge: The term “democrats” is associated with US Congress, Sonia with Indian National Congress, Internet with Mobile World Congress and so on.

Using Logistic for text classification problem

- A key issue to handle when using logistic or softmax for NLP applications is the arbitrary variable length of sentences.
- When we use the words in a sentence as features (as we did in the case of Naïve Bayes), the feature length n becomes a variable
- Logistic requires a fixed number of features and the associated theta parameters.
- Some options to consider to fit the input to a logistic:
  - Regardless of the size of the input sentence, always form an n-feature vector. If the sentence length is > n, chop off extra words and if it is < n, pad it suitably
  - Split a long sentence in fixed size blocks of n
- As we observe, such pre processing of input for the sake of fitting it in to a Logistic regression might degrade the prediction accuracy
- Solution: Use a different model ☺

MaxEnt Models

- MaxEnt models support an elegant way to map a variable length input in to a fixed size feature vector.
- This is a powerful model that has equivalence to logistic regression
- Many NLP problems can be reformulated as classification problems
  - E.g. Language Modelling, Tagging Problems (these will be covered in detail later)
- MaxEnt is widely used for various text processing tasks.
- Task is to estimate the probability of a class given the context
- The term context may refer to a single word or group of words
- The key idea is to use a bunch of “feature functions” that map the input in to a fixed length feature vector. Often these are Boolean functions.
- MaxEnt models are also termed as log-linear models

MaxEnt principle

Consider the *Principle of Maximum Entropy* [Jaynes, 1957, Good, 1963], which states that the correct distribution $p(a, b)$ is that which maximizes entropy, or "uncertainty", subject to the constraints, which represent "evidence", i.e., the facts known to the experimenter. [Jaynes, 1957] discusses its advantages:

> ...in making inferences on the basis of partial information we must use that probability distribution which has maximum entropy subject to whatever is known. This is the only unbiased assignment we can make; to use any other would amount to arbitrary assumption of information which by hypothesis we do not have.

More explicitly, if $\mathcal{A}$ denotes the set of possible classes, and $\mathcal{B}$ denotes the set of possible contexts, p should maximize the entropy

$$H(p) = -\sum_{x \in \mathcal{E}} p(x) \log p(x)$$

where $x = (a, b)$, $a \in \mathcal{A}$, $b \in \mathcal{B}$, and $\mathcal{E} = \mathcal{A} \times \mathcal{B}$, and should remain consistent with the evidence, or "partial information". The representation of the evidence, discussed below, then determines the form of p.

Representing Evidence

- One way to represent evidence is to encode useful facts as features and to impose constraints on the values of those feature expectations
- A feature is a binary valued function (indicator function):

$$f_i : \varepsilon \longrightarrow \{0, 1\}$$

- Given k features the constraints have the form:
  - Expectation value of the model for the feature f_j = Observed Expectation value for the feature f_j
  - $\sum_{x \in \epsilon} p(x) f_j(x) = \sum_{x \in \epsilon} \tilde{p}(x) f_j(x)$
- The principle of maximum entropy requires:

$$\begin{aligned} P &= \{p \mid E_p f_j = E_{\tilde{p}} f_j, \ j = \{1 \ldots k\}\} \\ p^* &= \arg\max_{p \in P} H(p) \end{aligned}$$

The general problem

- We have an input domain X
  - For example: A sequence of words
- There is a finite label set Y
  - For example: The space of all possible words – that is the vocabulary
- Our goal is to determine P(y|x) for any x, y where x is in the input space and y is in the space of labels
  - For example: Given an input sentence (that is x, a sequence of words), determine the next word in the sequence - that is $P(w_i \mid w_1..w_n)$

Feature Vector

- A feature is a function $f_k(x, y) \in \mathbb{R}$
- Often the features used in Log-linear models for typical NLP applications are binary functions that are also called indicator functions: $f_k(x, y) \in \{0, 1\}$
- If we have m features then a feature vector $f(x, y) \in \mathbb{R}^m$
- The number and choice of features for a given input is arbitrary. The system developer can design these with an intuition of the problem space he is addressing.

Features in Log-Linear Models

- Features are pieces of elementary pieces of evidence that link aspects of what we observe x with a label y that we want to predict (Ref: C Manning)
- A feature is a function with a bounded real value

$$f: X * Y \rightarrow \mathbb{R}$$

- Example:
  - Consider a sentence: "**Gandhi was born on 2 October 1869 in Porbandar**"
  - $f_1(x, y)$ = [y = PERSON and w_i = isCapitalized and w_{i+1} = ("was" | "is") and w_{i+2} = VERB]
  - $f_2(x, y)$ = [y = LOCATION and w_i = isCapitalized and w_{i+1} = ("was" | "is") and w_{i+2} = VERB]
  - $f_3(x, y)$ = [y = DATE and w_i = CD and w_{i-1} = ("on") and w_{i-2} = VERB]

Feature Based Models

- Many NLP problems can be formulated and addressed using feature based models.

- Text Categorization
  - Given some text assign a category label: Many words to one category

- Word Sense Disambiguation
  - Given a text and a subset of words that need to be disambiguated, assign word sense labels to each of the ambiguous words

- Tagging Problems
  - Given a text, assign a tag to each of the word tokens (e.g. POS tagging)

Other Examples

- Sentence Boundary Detection
  - A good amount of work has been done on performing this. However it will be interesting to look at this for informal text (e.g. tweets) and text written incorrectly.
- Sentiment Analysis
- Prepositional Phrase Attachment
  - Attach to verb phrase or noun?
- Parsing decisions
- Dialogue Systems

Feature functions for Twitter Search Case Study

- What are the words that suggest the model that a given tweet pertains to mobile world congress as opposed to congress party?

  E.g. internet, world, mobile, Samsung, Android, sensor, features, etc

```
def f1(tweet, y):
    if “internet” in tweet and y == “mobile_world_congress”:
        return 1
    else:
        return 0
```

- We can use other features such as the place of origin of the tweet

- The number and choice of features are entirely arbitrary and hence we can design a powerful feature set with some domain expertise

Parameter Vector

- Given the feature vector f(x, y) $\in \mathbb{R}^m$ we can define the parameter vector v $\in \mathbb{R}^m$

- Each (x, y) is mapped to a score which is the dot product of the parameter vector and the feature vector:

$$v.f(x,y) = \sum_{k=1}^{m} v_k f_k$$

Log-linear model - definition

- Let the Input domain X and label space Y
- Our goal is to determine P(y|x)
- A feature is a function: $f: X \times Y \rightarrow \mathbb{R}$
- We have m features that constitute a feature vector: $f(x, y) \in \mathbb{R}^m$
- We also have the parameter vector: $v \in \mathbb{R}^m$
- We define the log-linear model as:

$$\boldsymbol{p(y|x; v) = \frac{e^{v.f(x,y)}}{\sum_{y' \in Y} e^{v.f(x,y')}}}$$

Why log-linear?

- As before, we need to determine the model parameters using the training data.

- The log-likelihood of the MaxEnt hypothesis is given by:

$$\log(P(y|x;v)) = v.f(x,y) - \log \sum_{y'} \exp(v.f(x,y')$$

- The above has a linear term and a normalization term. Hence the name log linear classifier.

Parameter Estimation

We need to maximize:

$$L(v) = \sum_{i=1}^{n} v.f\left(x^i, y^i\right) - \sum_{i=1}^{n} \log \sum_{y'} \exp(v.f(x^i, y'))$$

Parameter Estimation: Gradient computation

$$\frac{\partial L(v)}{\partial v_k} = \sum_{i=1}^{n} f_k(x^i, y^i) - \sum_{i=1}^{n} \frac{\sum_{y'} f_k(x^i, y') \exp(v.f(x^i, y')}{\sum_{z'} \exp(v.f(x^i, z')}$$

$$= \sum_{i=1}^{n} f_k(x^i, y^i) - \sum_{i=1}^{n} \sum_{y' \in Y} f_k(x^i, y') \frac{\exp(v.f(x^i, y')}{\sum_{z'} \exp(v.f(x^i, z')}$$

$$= \sum_{i=1}^{n} f_k(x^i, y^i) - \sum_{i=1}^{n} \sum_{y' \in Y} f_k(x^i, y') \, \boldsymbol{p}(\boldsymbol{y'} | \boldsymbol{x^i}; \boldsymbol{v})$$

The algorithm for parameter updates is similar to earlier ones

Feature and Model expectations

$$\sum_{i=1}^{n} f_k(x^i, y^i) - \sum_{i=1}^{n} \sum_{y' \in Y} f_k(x^i, y') \, \boldsymbol{p}(\boldsymbol{y'} | \boldsymbol{x^i}; \boldsymbol{v})$$

- We observe that the above expression that represents the optimization criteria has 2 terms that correspond to:
  - Feature expectations (also called empirical counts, seen from the data)
  - Model expectations (produced by the model)

- Maximizing the likelihood implies that the model should exactly account for the evidence seen in the training data

- Note that maxent is a logistic regression and we can determine the point of minimum cost using suitable numerical techniques

Summary

- Linear models assume a linear decision boundary and are simple models

- Though the models are simple, they are extensively used as:
  - Stand alone classifiers
  - As a layer or a building block in multi layer models (such as Neural Networks, CNNs, RNNs)

- MaxEnt models are widely used in NLP applications