Logistic Regression

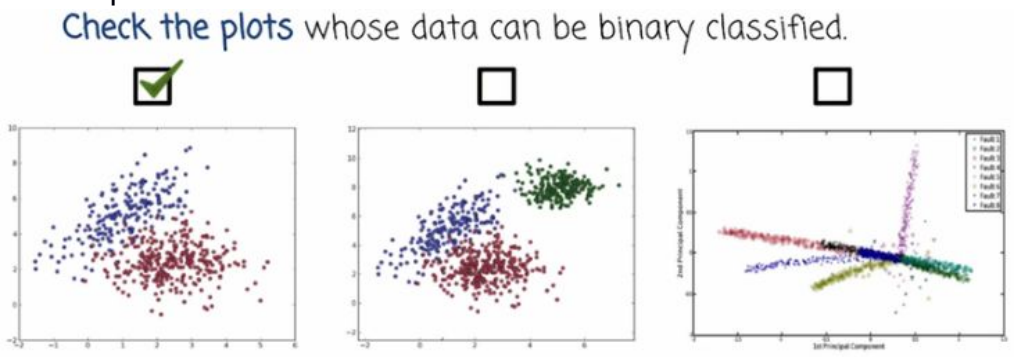
Notes for CS 6232: Data Analysis and Visualization
Georgia Tech (Dr. Guy Lebanon), Fall 2016
as recorded by Brent Wagenseller

Lesson Preview

- Classification is the most common problem in machine learning
- **Classification** is predicting a label associated with a vector of measurements
  - Critical to spam or fraud detection, click prediction, newsfeed ranking, digital advertisement, etc
- We will learn the theory of **logistic regression** (the most common classifier) and how to use it in practice

Logistic Regression

- When is it a good time to use Logistic Regression?
  - When there is a binary (or **nominal**) outcome
    - Example: User either clicks or does not click on a website
    - Basically two possible outcomes (usually yes or no)
      - Note that this is not multiclass classification; binary classification strictly deals with 2 choices at most
      - Example



        - The first is a binary classification, the others are a multiclass classification
  - There is one or more measureable variable
    - These are used to construct a prediction
  - When predictions about the nominal variable can be made
- Logistical Regression does have multiclass generalizations, but we will not cover it in this module
- Other examples of binary classification
  - Ad placement
  - Feed ranking
  - Recommendation Systems
  - Spam detection / filtering
  - Credit card fraudulent transaction
  - Medical testing (to determine illness)

Definitions

- We will have a vector of measurement variables 'x' as such:
  $x = (x_1, x_2, x_3, x_4, \ldots. x_d)$
  - This is known as the **measurement vector**, and it describes the measurements
  - Each individual x_d is known as a **feature**
  - This is a vector of 'd' components

- We will note it as ‘x’, but understand it can (and most likely will be) a vector
- Each ‘x’ vector will be noted as $x^{(i)}$, which indicates the entire vector (in almost every case there will be multiple x vectors)
  - In effect: $x^{(i)} = (x^{(i)}_1, x^{(i)}_2, x^{(i)}_3, x^{(i)}_4, \ldots. x^{(i)}_d)$
  - The subscript will be reserved for the component of the vector, and the superscript will refer to an entire individual vector x

- The response variable will be y
  - ‘y’ will either be
    - 1= ‘yes’
    - -1= ‘no’
      - NOTE: This can sometimes be 0
  - ‘y’ is the variable we are trying to predict
  - This is also known as the **label**
  - Much like x, y is also denoted as $y^{(i)}$ and will correspond to vector $x^{(i)}$
- Theta (Θ)
  - **Theta** (**Θ**) is a vector that describes the classifier
  - This is a vector of parameters (also known as the **parameter vector**)
  - It has d components, which will match the d components in the x vector
  - The inner product seems to be important, particularly between Θ and x
    - $\langle x, \Theta \rangle = \Theta_1 x_1 + \Theta_2 x_2 + \Theta_3 x_3 + \ldots + \Theta_d x_d$
      - Basically, sum these to get the inner product
  - Θ is also known as the weights vector
    - As in, feature x_1 should have a weight of .05, x_2 has a weight of .20, x_3 has a weight of .1 … x_d has a weight of .3
    - From the quiz ‘predicted classes’ it seems these do not need to be less than 1 nor equal 1
    - Also, according to the quiz ‘predicted classes’
      - A positive Θ (or weight) assumes an increase in its respective feature to influence a y that will =+1
      - A negative Θ (or weight) assumes a decrease in its respective feature to influence a y that will =-1
      - BRENTS NOTE: this may have been specific to the quiz, as if the feature in question is negative it would have the obvious opposite effect
- The task of classification is given a vector of features x, assign a label y
  - The goal here is to take a bunch of historical data (training data / a training set of data) that has x features with a y classification, and then devise a mathematical method to determine
  - This is done based on a training set
    - A training set consists of pairs of measurement vectors and response
      - Example: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots (x^{(n)}, y^{(n)})$
      - Note the number of pairs is denoted as ‘n’
    - This is gathered from historic data
  - **Training data** is a table (dataframe) of rows representing training set examples and labels
    - Example

	User Characteristics				
	x_1	x_2	x_3	x_4	y
					click
					no click

    - This is an example of ML applied to if a user will click on an ad in LinkedIn
    - X = vector of features related to the specific user

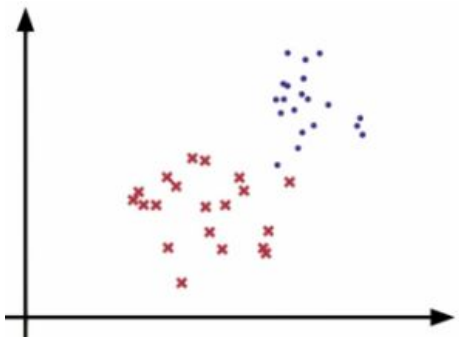
        - These can be specific words, phrases, or topics
        - Can be length
        - Can be the relationship of the person who posts the feed to the user who views the feed (or their interaction counts)
    - Y = the user clicks or does not click

Linear Classifiers

- Linear classifiers have a prediction rule that have the following algebraic form:
  - $Y = \text{sign}(\langle \Theta, x \rangle)$
    - $\langle \Theta, x \rangle$ is the inner product
    - Θ is the parameter vector describing the classifier; it's the same for all x vectors!
  - The sign function works as follows
    - If the dot product of $\langle \Theta, x \rangle$ is positive, Y is assigned to be '+1'
    - If the dot product of $\langle \Theta, x \rangle$ is negative, Y is assigned to be '-1'
    - If the dot product of $\langle \Theta, x \rangle$ is zero, Y can be either positive or negative
- Why linear classifiers?
  - They are easy to train
  - The prediction can be very quick
    - They predict labels very quickly at serve (query) time
    - These can be parallelized (if the dimensioning is high) so different processors can hold/compute subparts of the inner product
    - These can also be fast if the parameter vector OR the measurement vector is mostly zeros
      - The vector that is mostly zeros is called a **sparse vectors**
    - In many industries, the prediction time is far more important than the training time; query speed is usually crucial
  - They have a well established statistical theory behind it; this well known statistical theory of linear classifiers leads to effective modeling strategies
    - Helps us understand which classifiers to use and which to not use
  - Linear classifiers are particularly useful in high dimensions (a high 'd' value); they excel at high dimensions due to
    - Their simplicity
    - Attractive computational load
      - Even with millions of features
    - Nice statistical properties
  - For the visualization of data, assume dimension = 2
    - Typically its much higher

The Linear Plane

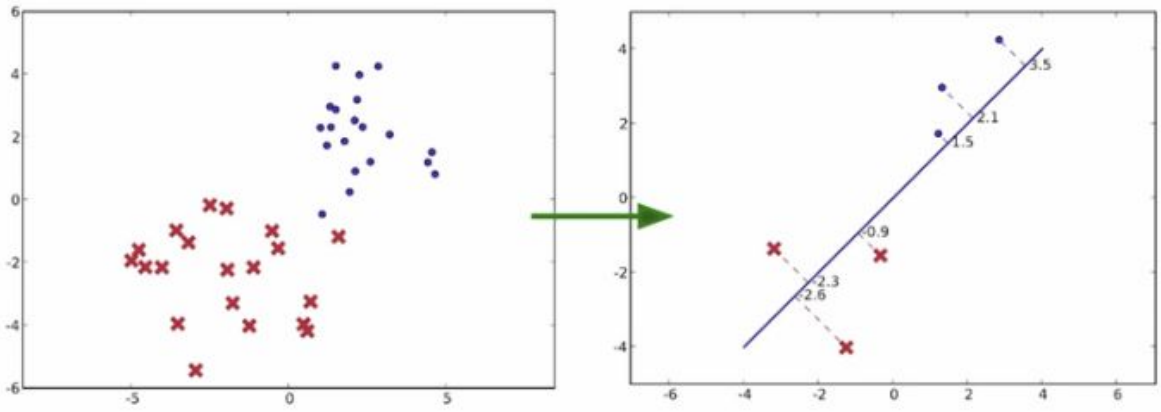
- Linear classifiers are called linear because the **decision boundary** - the area that distinguishes between what is positive and what is predicted as negative - will be a flat shape
  - In 2D, it's a line
  - In 3D, it's a 2D plane
  - In the general case, it's a d-1 dimensional hyperplane
  - Example



- Given a vector of two dimensions: $x = (x_1, x_2)$
  - The classification will be: $\text{sign}(ax_1 + bx_2 + c)$
    - Or, using theta: $\text{sign}(\Theta_1 * x_1 + \Theta_2 * x_2 + \Theta_3)$
      - C is what is known as an offset term
      - Note that technically, c (the last Θ) is not represented in the x vector; a workaround is to pad the x vector with a 1 at the end so the last Θ (the constant) can be used
  - For classification of the above data:

Sign($ax_1 + bx_2 + c$)	Classification is
positive	+1
negative	-1

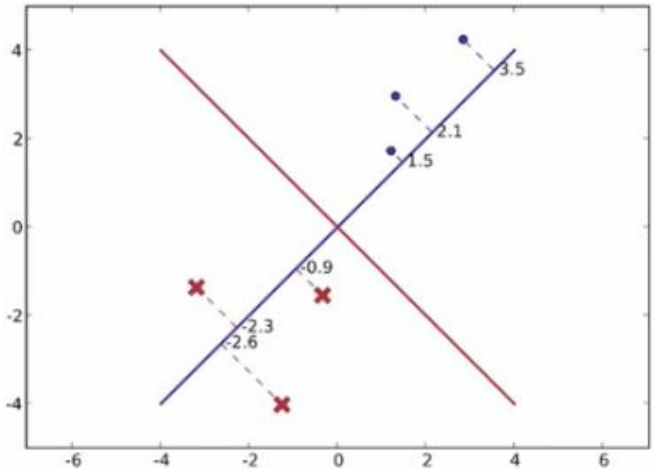
- How the linear classifier works



  - When we take an inner product of these two datasets (in the graph) between the vector of measurements and the weight vector - IF the weight vector is **normalized** (meaning it has unit length to be of length 1) - the outcome of the unit product is basically the projection of the input vector into that unit (weight) vector
    - More on the normalization
      - In a quiz given in the beginning of the lecture (and referenced above), Dr. Lebanon used weights that were clearly not normalized to 1
      - The classification decision does not matter whether you renormalize theta or not
      - That said, if you want to see the geometric interpretation, its easier to see that if the theta vector is normalized
      - The decision boundary (see below) will not change whether we multiply theta by a constant positive scalar or not (we will get the same decision boundary and the same prediction rule)
    - The unit vector is the theta vector that corresponds or describes that projection direction
    - The corresponding inner product would capture the distance of the point from the

hyperplane (in the second graph) which will correspond to the decision boundary

- The **decision boundary** is the red line and shows the demark point between the two datasets



  - The decision boundary is also known as the **hyperplane**
    - It's a hyperplane instead of a line as it's a line in 2D, but it will be a 2D plane in 3D etc etc etc
    - This decision plane / hyperplane describes the classifier
  - The **theta vector** is perpendicular (aka **orthogonal**) to the decision boundary (as seen above)
  - The projection of each point onto the theta vector shows how far away it is from the decision boundary
    - To see this, look at the graph above; the values of the inner product for select points are shown with their value
    - BRENTS NOTE: It seems as if the intersection of the decision boundary and the theta vector are situated to pass through the origin
      - That said, Dr. Lebanon spoke to this in a quiz
        - It does not have to go through the origin. If we do not require this (by using the trick that involves padding the x vector with 1s) the classifier becomes considerably more powerful

- Prediction is only the first part – we also want to predict some confidence or probability in our prediction rule
  - We need to map the inner product into a confidence / probability
  - One function that can do this is the sigmoid function
  - A sigmoid function is one way of getting a probabilistic classifier that doesn't just predict whether a label is +1 or -1; it also predicts a probability associated with that fact
    - Knowing this probability can be helpful, as it can help us rank things
      - for example, ad space on a webpage – we can put the 'most likely' links to be clicked first

Bias Term

- The **bias term** is 'c' which was mentioned earlier
  - This was mentioned in the discussion regarding padding the x vector with a 1 at the end; basically, it's a theta with no organic corresponding x value
- The bias term is useful as this is what allows the decision boundary to not be required to pass through the origin

- Recall that the trick to adding the bias term is adding the bias term as the 'last' theta and then adding a 1 to the end of the measurement vector (x vector)
- This trick becomes critical later, as there is some math involved that needs the bias term; if this trick is not done those math problems become more complicated

Increasing Data Dimensionality

- The biggest drawback of linear classifiers is the decision boundary; its quite possible the data cannot be divided by a hyperplane (that is to say, the points are intermingled)
  - There is a trick to overcome this while keeping all of the benefits of a linear classifier (simplicity, scalability, etc)
  -
- For example, assume we have a two dimensional data vector; convert it to a six dimensional vector!

$$x=(x_1,x_2) \rightarrow \hat{x}=(1,x_1,x_2,x_1^2,x_2^2,x_1x_2)$$

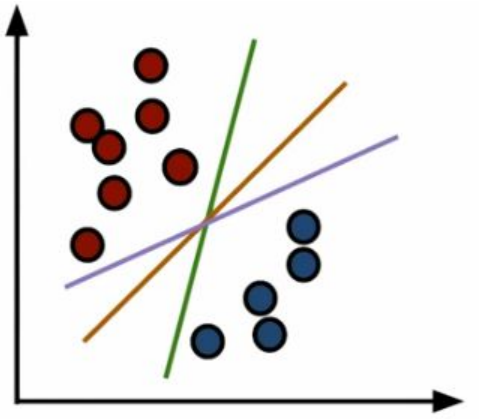
  - We transform x to x(hat) which has 6 dimensions
  - In the 6 dimensional space, the decision vector will now be hyperplaned
    - In the original space the decision boundary could be highly nonlinear, but using this trick helps mitigate that
    - This helps us capture nonlinear trends when we use linear classifiers
- Increasing the dimensions is worth noting as now there are more computations and more storage is needed
  - That said, we can work with this increase so long as the parameter or feature vectors are mostly zero
- It should also be noted that this increases the dimensionality of theta
  - This may lead to overfitting if we do not have enough data in our training set!
- In summary
  - Its possible to transform the data from x to x(hat), where the classifiers (y) stays the same
    - The original data must be transformed to the transformed vector; this also means data we wish to make predictions from must also be transformed

Classifiers

- Classifiers define a map from a vector of features x to a label
  - Sometimes we get a confidence, and sometimes that confidence is also the probability that the label is 1
  - Probabilistic classifiers provide that tool by defining the probabilities of the labels +1 and -1 given the features of vector x
- We know from probability theory that
  - $p(y = +1|x) + (p(y=-1|x) = 1$
    - That is to say, the probability of y being +1 (given x) and y being -1 (given x) will equal 1
  - Because of this, the probabilistic classifier should give us a way of either measuring the first OR second value
    - The other value we can just get by subtracting the first number from 1
- The probability that a given element of vector x will be classified as 1:
  - $P_\Theta(Y=1 \mid X=x)$
- The probability that a given element of vector x will be classified as -1:
  - $P_\Theta(Y=-1 \mid X=x)$

Maximum Likelihood Estimator (MLE)

- The **Maximum Likelihood Estimator** (**MLE**) is one of the most well-known methods of training probabilistic classifiers
- Start with a vector of points in a Euclidian space (same as the other x measurement vector)
- Example



  - The red and blue points are the two different classifications
  - The three different lines are potential different classifiers / decision boundaries
    - All of them have different prediction probabilities
- Maximum likelihood assumes a specific mapping from theta to the probability that Y = 1 (or -1)
  - We will use logistic regression as this mapping
- The thinking behind the MLE: Give me a hyperplane that will maximize the likelihood of the data
  - aka, best explains the data
- If we had a hyperplane that explained the data very well (in other words, maximizes the likelihood of the data) this will probably be a good classifier of the data
- Two statistical philosophies:
  - Frequentists philosophy
  - Bayesian philosophy
- MLE is an example of **frequentists** philosophy
  - The **frequentists philosophy** is a philosophy in statistics that talks about nature being something that we are trying to predict
    - There is one correct state of nature that we do not know
    - The task of the statistical procedure is to predict that state of nature using observations
  - In other words…
    - Frequentists Maximum Likelihood Estimator uses pairs of feature vectors and labels $((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots (x^{(n)}, y^{(n)}))$ usually from historic data to estimate correct classifier or nature
  - Frequentists philosophy is not bound to only the MLE
  - <u>Note that we will focus on the MLE for this course as it is the dominant philosophy in the industry at the moment</u>
- Bayesian statistics is an alternative to MLE and frequentists philosophy
  - Bayesian statistics claims that a single classifier cannot represent the ‘truth’. Estimate the revised probability that each classifier is correct and use them all
  - In other words
    - There is no single correct state of nature that generates the data
    - Therefore, there is no point in thinking about a specific model that generates data that we will try to predict as there is a collection of different models, with each model being correct with some probability
      - We then use all of the models together to inform our decision making
  - <u>We will not use Bayesian much in this course, but just know it exists</u>

- Comparing both philosophies
  - Frequentists philosophy
    - 'traditional' philosophy
    - The industry still heavily relies on Maximum Likelihood Estimators
  - Bayesian philosophy
    - Gaining ground lately
    - ML is starting to rely more on Bayesian
    - Takes more resources and time to produce (when compared to MLE)

Maximum Likelihood Estimator (MLE) Defined

- Equation:

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \quad p_{\theta}(Y = y^{(1)}|X = x^{(1)}) \cdots p_{\theta}(Y = y^{(n)}|X = x^{(n)})$$

  - That is to say, the θ that maximizes the likelihood of the data (the argmax θ)
  - The likelihood of the data is the product of the conditional probabilities of the labels given the feature vectors
    - We take a product over all of the probabilities of the training data pairs
    - The product is a likelihood, and it is a function of θ (because θ describes the probabilistic classifier)
    - Because it's a function of θ, we can investigate which θ maximizes it and then take that maximizer as the classifier we will use (which is the MLE)
      - This θ we can use later in prediction/query time so we can classify new vectors / data
- We also can use the log likelihood

$$= \arg\max_{\theta} \quad \log p_{\theta}(Y = y^{(1)}|X = x^{(1)}) + \cdots + \log p_{\theta}(Y = y^{(n)}|X = x^{(n)})$$

  - As it turns out, this is useful
  - because log is a concave function, the maximizer of the product will be the same as the maximizer of the log of the product
    - The product and the log of the product of the maximum will be different, but the maximizer (the θ that maximizes the product and the θ that maximizes the log of the product) will be the same
      - In other words, the θ itself will not change; the value of the maximized result WILL change, but the θ used to create that maximum will not change
  - The reason we use the log is the log of a product is the sum of the logs
    - This simplifies the expression, <u>especially later when we want to compute derivatives</u>
    - <u>We will be using the log likelihood and not the likelihood as the log likelihood is far superior for the task at hand</u>
- The MLE is the maximizer of the likelihood (which is the same as the maximizer of the log likelihood)
- Justification for using the Maximum Likelihood Estimator
  - It converges to the optimal solution in the limit of large data (known as the **consistency property**)
    - Disclaimer: Data is generated based on the logistic regression model family and n (number of rows in training data) approaches infinity while d (number of columns) is fixed
  - The convergence occurs at the fastest possible rate of convergence (**statistical theory**)
    - Note that when the dimension count is high, some of these claims do not hold anymore as they assume d is fixed and n goes to infinity

        - In practice, d can be billions of features – greater than n even – which means this breaks down (recall the curse of dimensionality which claims you need a certain number of rows more than d)
- MLE Quiz
    - Question: Describe a computational procedure for teaching the value x for which f(x) is at a maximum. Does it scale to high dimensions of x?
    - Answer
        - Procedure
            - Compute f(x) on a grid of all possible values and find the maximum
            - Do this for scalars x or low dimensional vectors x
        - This does not scale to high dimensions. An alternative technique that does scale is gradient ascent)

Probabilistic Classifiers

- Logistic Regression is the most popular probabilistic classifier
- Logistic regression
    - Scales well both to high dimensions and large datasets
    - Is simple
    - Is popular
- Definition of Logistic Regression:

$$p(Y = y|X = x) = \frac{1}{1 + \exp(y\langle\theta, x\rangle)}$$

    - The probability of Y being either +1 or -1 (little y is the value that big Y (the random variable) takes) conditioned on a vector X taking on a specific value little x EQUALS
    - 1 / (1 + e^ y*< θ,x>)
        - Remember that y is either +1 or -1
        - < θ,x> is the inner product of θ and x
    - <u>This is the formula of the logistic regression classifier</u>
    - The only unknown is the vector θ
        - Once this is known, logistic regression will give you a probability for y equals +1 or -1 given any vector of measurements x
    - Dr. Lebanon did show that the two probabilities – one for y=+1 and one for y = -1 – do indeed sum to 1
        - Also, the probabilities will always be nonnegative
- Quiz: Decision Boundary
    - Question
        - Where should the decision boundary be placed
    - Answer
        - The set of points where p(Y= 1|x) = p(Y= -1|x) = .5

Review of Tools (Formerly: Prediction Confidence)

Task	Use
Predict the label associated with a feature vector x	prediction rule sign (θ, x)
Measure of confidence of that prediction	$p(Y = y \mid X = x) = 1/(1 + \exp(y\langle\theta, x\rangle))$

- If we simply want to predict the label, use the prediction label
- If we want a measure of confidence, use the logistic regression formula
  - Of course, this assumes you have figured out the vector θ
  - Not only can this give a confidence, this has a direct interpretation as probability

MLE and Iterative Optimization

- Our goal is to develop a procedure on how to maximize the likelihood
- Again, the formula for logistic regression:

$$p(Y = y|X = x) = \frac{1}{1 + \exp(y\langle\theta, x\rangle)}$$

- And again, the likelihood

$$\begin{aligned}\hat{\theta}_{\text{MLE}} &= \arg\max_{\theta} \quad p_\theta(Y = y^{(1)}|X = x^{(1)}) \cdots p_\theta(Y = y^{(n)}|X = x^{(n)}) \\ &= \arg\max_{\theta} \quad \log p_\theta(Y = y^{(1)}|X = x^{(1)}) + \cdots + \log p_\theta(Y = y^{(n)}|X = x^{(n)})\end{aligned}$$

  - Again, this is the product of the conditional probabilities of the labels given the feature vectors; this is what we want to maximize
  - Again, the set of θ that will satisfy the MLE will also satisfy the log MLE
- Combining them together

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \sum_{i=1}^{n} \log \frac{1}{1 + \exp(y^{(i)}\langle\theta, x^{(i)}\rangle)}$$

  - We are left with maximizing the sum of the log of the probabilities of the labels given their data points; we plug this in to the logistic regression formula and we are left with the above equation
  - That said, we can reduce this a bit to

$$\arg\max_{\theta} \sum_{i=1}^{n} -\log\left(1 + \exp(y^{(i)}\langle\theta, x^{(i)}\rangle)\right)$$

  - From math, we know that log(1/a) = -log(a); so we can switch this around
  - It's a bit odd to maximize on a negative number, so we can remove the negative and change the max to a min and use this equation instead:

$$\arg\min_{\theta} \sum_{i=1}^{n} \log\left(1 + \exp(y^{(i)}\langle\theta, x^{(i)}\rangle)\right)$$

    - This equation is important

Gradient Descent

- We still have to minimize an expression (to eventually find the vector θ)
- Gradient descent helps find the minimum of a function where the minimum of a function is computed for vectors that may grow to a higher dimensionality and the procedure is scalable
- Steps
  - Step A: Initialize the dimensions of the vector θ to random values
  - Step B: For j = 1…,d, update θ_j according to this rule:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial \sum_{i=1}^{n} \log\left(1+\exp(y^{(i)}\langle\theta, x^{(i)}\rangle)\right)}{\partial \theta_j}$$

    - Update θ_j to be the old value of θ_j minus alpha (which is some number called the **step size**, which may correspond to the learning rate) times the partial derivative of the log likelihood (or the negative log likelihood in this case) with respect to θ_j
    - The expression (everything that is not θ_j - α) is a function of the vector θ, and when we want to take the derivative with respect to θ_j (that's going to be a partial derivative) we are going to fix all of the other components and take the derivate only with respect to θ_j
      - This tells us how much we should decrement the current θ_j, and this is done for all dimensions
    - After working out the (partial) derivatives, this is the final equation that must be used:

$$\Theta_j = \Theta_j - \alpha\left(\sum_{i=1}^{n} \frac{\exp(y^{(i)}\langle\theta, x^{(i)}\rangle) \cdot y^{(i)} x_j^{(i)}}{1+\exp(y^{(i)}\langle\theta, x^{(i)}\rangle)}\right)$$

      - THIS IS THE EQUATION TO BE USED IN WRITING YOUR ALGORITHM
      - A special thanks to Rafael Espericueta from the Fall 2016 semester for figuring this out!
      - Please see Rafael's detailed, step-by-step explanation of this. I didn't attach it here but it is on the forums in post @527; you should have a working understanding of how this was done
  - Step C: Repeat the update (step B) until the update becomes smaller than a pre-determined threshold
    - This threshold is either A) a delta value threshold or B) a time threshold
    - Let alpha decay as the gradient descent iterations increase
      - Dr. Lebanon mentions that this could be the square root of the number of iterations
        - This may not work if you are considering alpha to max out at 1
    - If we choose alpha correctly and our delta to be sufficiently small, this will converge!
- The reason this works – and why its called gradient descent – is because step B basically means that the vector θ is adjusted with the gradient
  - The gradient is the vector of partial derivatives
  - If we wanted to write step B in vector form, it would be vector θ is the old vector θ minus alpha times the gradient
    - The gradient (from calculus) is the direction of steepest ascent
    - Here we have minus (Dr. Lebanon points to the upside down delta), so the minus of the gradient would be the direction of steepest descent
    - The direction at which, if we were to follow, would 'go down' the fastest
    - If we keep following the negative gradient, we will keep going down until we hit a local minimum
      - In our case, because we are minimizing a negative log likelihood, we will maximize

the log likelihood and reach a hilltop that will have a gradient of 0 to indicate that it is a local maximum

- Gradient descent is easy to compute and scales well
- Review
  - This is the minus of the log of the probability of the label given the training point

$$\log\left(1+\exp(y^{(i)}\langle\theta,x^{(i)}\rangle)\right)$$

  - We then take the sum of all such logs:

$$\sum_{i=1}^{n}\log\left(1+\exp(y^{(i)}\langle\theta,x^{(i)}\rangle)\right)$$

  - Then we want to minimize that because this is minus of the log of probability:

$$\arg\min_{\theta}\sum_{i=1}^{n}\log\left(1+\exp(y^{(i)}\langle\theta,x^{(i)}\rangle)\right)$$

  - In the case of linear regression, the likelihood function is concave, meaning there is no more than 1 local maximum
    - If we use gradient descent, we will slowly (or quickly) converge to the maximizer of the likelihood, regardless of where we start
      - We will NOT get caught in local maxima
      - That said, there are situations where the maximum is at θ_j as it approaches infinity
        - Usually if this happens we are overfitting
        - When this happens, terminate and use what we have

Stochastic Gradient Descent

- Usually gradient descent scales well, but sometimes for very large data sets its slow
- To combat this, we can use stochastic gradient descent
  - The difficulty with gradient descent is that for every iteration, you have to cycle through every single data point
    - In contrast, stochastic gradient descent gives small updates continuously to the vector θ as opposed to waiting to give an update to vector θ until a full loop is complete
      - We see updates to vector θ faster
      - By the time we do a complete loop over the entire training set, we have already updated theta many times
      - This takes a small step after seeing every data point, as opposed to gradient descent which, to take one step, it needs to loop through the entire dataset once
- Steps for Stochastic Gradient Descent
  - Step A: Initialize the dimensions of the vector θ to random values
  - Step B:
    - Instead of the old step B, pick one labeled data vector ($x^{(i)}$, $y^{(i)}$) randomly, and update each dimension

$$j=1,\ldots,d:\ \theta_i\leftarrow\theta_j-\alpha\frac{\partial\log\left(1+\exp(y^{(i)}\langle\theta,x^{(i)}\rangle)\right)}{\partial\theta_j}$$

        - This is kind of like a gradient descent step, but based on one single example
        - This is computing the log likelihood and taking the gradient of the negative log likelihood based on the single example and update all dimensions
        - The computational load is much lighter as its based on a single data point
    - Step C: Repeat step B (keep picking random rows) until the updates of the dimensions become too small (reducing alpha as the number of iterations increases)
- A benefit is this can be done across multiple machines (say, in a cloud)

Overfitting

- A good model selection may not be perfect on the training data, but it generalizes to new data
    - Sometimes things will just be miss-classified; noise will happen
    - If we try to perfectly classify the data, the model (the vector θ) may be too tuned to the training data and will not generalize well to data 'in the wild'
        - This is called **overfitting**, where we try to fit the model to the training data perfectly but the model does poorly when trying to predict data that is not in the training set
            - In other words, overfitting is fitting a model too aggressively to the training data to the point where it may not generalize well
            - In other other words, there are too many parameters to estimate from the available labeled data
            - In essence, we train the model to random noise that exists in the data
- When x is high dimensional overfitting becomes a problem
    - This is particularly true when compared to the size of the training set
        - When the dimensionality is high and the number of training samples is low, overfitting is more likely to happen
- How to combat overfitting
    - Choose a classifier that is more resistant to overfitting
    - Linear classifiers are relatively resistant to overfitting
        - That said, adding nonlinear transformations of the original features – as described earlier in the lesson to avoid the constraint of having a linear decision boundary – can cause overfitting
    - Feature selection can work – eliminate the features that are not as helpful / remove irrelevant features
    - Use Regularization (see next section)

Regularization

- **Regularization** refers to adding another penalty term to the maximum likelihood cost function
- Regularization can help mitigate overfitting
- The **'L1' Penalty**

$$\beta|\theta_1| + \cdots + \beta|\theta_d|$$

    - Beta times the absolute values of the different components
- The **'L2' Penalty**

$$\beta\theta_1^2 + \cdots + \beta\theta_d^2$$

    - Beta is some number that multiplies the squares of the components of the feature vector θ
- L1 and L2 are two of the most common regularization
    - Both of them basically penalize high values of θ
    - When we think about maximizing the likelihood, remember this gets converted to minimizing the negative log likelihood

        - We want to minimize this cost function
        - We add these penalty terms, which prevent the theta from becoming bigger
- Think of this as telling the likelihood function ‘be careful with your likelihood maximizer; if it has high values of θ, it will be penalized
    - This means that these values will be very good at fitting the data as it has to overcome the penalty
- Selecting β
    - Its not easy to select β
    - B is selected through experimentation
    - B should be close to the optimal value
- When we add these penalty terms, we prevent the optimal θ from being achieved at infinity
    - As we increase θ, the penalties become larger
    - This can be a benefit sometimes as it forces the maximizer from going to infinity

Lesson Summary

- We learned about
    - Linear classifiers
    - Probabilistic Classifiers
        - How to train these using the Maximum Likelihood paradigm
    - Logistic Regression
        - How to use this in practice