**Machine Learning**

**Session 4**

1. **Supervised Learning**: Applications where the training data comprises examples of input vectors along with corresponding target vectors
   1. *Regression*: Desired output consists of one or more continuous variables
   2. *Classification*: Desired output consists of a finite number of discrete categories.
2. **Linear classifier:**
   1. Goal:
      1. Take an input vector x and assign it to one of K discrete classes y.
   2. Assume an partition of the feature space: y=f(x)
   3. Given examples (x\_i, y\_i), which may be noisy
   4. Learn f(x), to enable prediction of y\* given new point x\*. It should generalise well to new x\* – E.g., x1 : Fish Weight, x2 =Fish Length, y=Fish Species.
3. **Linear vs. Logistic Regression**:
   1. Linear & Logistic Regression use different representation/model assumptions:
      1. Linear Regression: straight graph represented by f\_w(x) = w^T x.
      2. Logistic Regression: characteristic S-shaped curve around [0, 0]; represented by f\_w(x) = 1 / (1 + exp^{-w^T x}).  
           
         Derived from the sigmoid logistic function:  
         sigma(x) = 1 / (1 + exp^{-x})  
           
         All sigmoid functions have the property that they map the entire number line into a small range such as between 0 and 1, or -1 and 1, so one use of a sigmoid function is to convert a real value into one that can be interpreted as a probability.
      3. What if x goes to -/+ infinity? (Assume w=1)
         1. x controls range of sigma(x) in [0,1].
      4. What if x=2 and we increase/decrease w magnitude?
         1. w Controls slope of f\_w(x)
   2. Modelling classes:
      1. Using assumption: f\_w(x) = 1 / (1 + exp^{-w^T x})
      2. Probability model:  
           
         p(y given x, w) = Bernoulli(y given sigma(w^T x)).
         1. Probability of class y is sigma(w^T x).
   3. Logistic Regression:  
        
      p(y given x, w) = sigma(w^T x) / (1 + exp^{-w^T x}). With range from 0, 1  
        
      P(Class 1 given x) = sigma(w^T x) --> sigma(w^T x) > 0.5 : class 1   
      P(Class 1 given x) = 1 - sigma(w^T x) --> sigma(w^T x) < 0.5 : class 0
   4. **Decision boundary**: Boundary between two classes. Contour where:  
        
      p(Class 1 given x) = p(Class 0 given x) = 0.5

1 / (1 + exp^{-w^T x}) = 0.5  
  
2 = 1 + exp^{-w^T x})

0 = w^T x (which is a straight-line function)  
  
If you have two clusters of points on different sides of a xy graph, parameters w specifies exactly which straight line we try to separate the data with.

* 1. **Unpacking the decision boundary**:   
       
     Really we should be doing:  
       
     sigma(w^T x + w\_0) / (1 + exp^{-w^T x + w\_0}).  
     1. Recall in linear regression, we used the trick x’ = the interval [1, x]
     2. Now w’ includes an interval [w\_0, w].
     3. w’^T x’ = w^T x + w\_0
  2. A 3D graph with three vectors originating from the origin [0, 0, 0]. The decision boundary is perpendicular to the vector w.
     1. Weight w\_0 controls the displacement of the line from origin.

1. **Objective function**:
   1. Recall from linear regression: f\_w(x)= w^T x --> E(w) = summation of (yi - w^T x)^{2} [Square deviation of prediction from label].
   2. For logistic regression:   
      f\_w(x) = 1 / (1 + exp^{-w^T x}) --> E(w) = negative product sum of {(yi times log p(y=1 given x\_i) + (1 – y\_i)log(1 - p(y=1 given x\_i))}
2. **Learning**: Find the weights w that minimize the cost function for data {y, X} or {(y\_i,x\_i)}:  
     
   p(y=1 given x) = sigma(w^T x) = 1 / (1 + exp^{-w^T x}) and,   
   E(w) = negative product sum of {(yi times log p(y=1 given x\_i) + (1 – y\_i)log(1 - p(y=1 given x\_i))}  
   1. Taking derivatives w.r.t weights:

dE(w)/dw = negative summation of (sigma(w^T x) – y\_i) times x\_i  
dE(w)/dw = -X^T (p(y given X) – y)

* 1. **Convex vs closed form**:  
       
     Find the weights w to minimize the cost function for data {y, X} or {y\_i, x\_i}:
     1. dE(w)/dw = negative summation of (sigma(w^T x) – y\_i) times x\_i  
          
        This time (unlike linear regression), we can’t rearrange to solve for w! It’s stuck inside the sigmoid.  
          
        No closed form solution. Only gradient. However it is convex. Remember what convex means?  
          
        Gradient will get the solution as unique minimum.
  2. Find the weights w that minimize the cost function for data {y, X} or {y\_i, x\_i} according to}:
     1. E(w) = negative product sum of {(yi times log p(y=1 given x\_i) + (1 – y\_i)log(1 - p(y=1 given x\_i))}
     2. Algorithm repeats until convergence:
        1. w' = w + alpha(E(w) / dw) which is equal to   
           w' = w – alpha times the summation of sigma(w^T x) – y\_i) times x\_i.

1. **Checkpoint**:
   1. Check you understand the dimensions of any algorithm/linear algebra expression.
   2. Questions:
      1. If input data is d dimensions, how many dimensions is w? x?
      2. How many dimensions is sigma(w^T x)?
      3. How many dimensions is E(w)?
      4. How many dimensions is dE(w)/dw?  
           
         p(y=1 given x) = sigma(w^T x) = 1 / (1 + exp^{-w^T x}) and,   
           
         E(w) = negative product sum of {(yi times log p(y=1 given x\_i) + (1 – y\_i)log(1 - p(y=1 given x\_i))} and,   
           
         dE(w)/dw = negative summation of (sigma(w^T x) – y\_i) times x\_i
2. **Confidence**: Larger magnitude weights result to steeper sigmoid.   
     
   p(Y given X,w) = the product sum of (p(y=1 given x\_i)^{y\_i} times (1 – p(y=1 given x\_i))^{1 – y\_i}) and,   
   p(y=1 given x)) = sigma(w^T x) = 1 / (1 + exp^{-w^T x}))  
     
   Graphs with different elongated S shapes.  
   1. If w=1. sigma(w^T x\_i)=[0.05,0.9,0.9]
      1. Probability of all data = 0.05\*0.9\*0.9 = 0.04
   2. If w=inf. sigma(w^T x\_i)=[0,1,1]
      1. Probability of all data =0\*1\*1 = 0
3. If w=1. sigma(w^T x\_i)=[0.9,0.9]
   1. Probability of all data = 0.9\*0.9 = 0.8
4. If w=inf. sigma(w^T x\_i)=[1,1]
   1. Probability of all data =1\*1 =1 (Learning will increase weight forever).

Larger weights => more confident prediction.

1. **Regularized Logistic Regression:**
   1. To address overconfidence, we can add an L2 norm regularizer (addition of an extra term to the learning equation).  
        
      E(w) = negative product sum of {(yi times log p(y=1 given x\_i) + (1 – y\_i)log(1 - p(y=1 given x\_i))} + lambda w^{T} w.
   2. *L2 norm regularizer*: A normal gives a scalar magnitude from a vector.  
        
      w\_0 = sum of w\_k (L0 norm)  
      w\_1 = sum of positive w\_k (L1 norm – Manhattan)  
      w\_2 = square root of the squared sum of w\_k (L2 norm – Euclidean)
   3. Cost is scalar.
      1. Need a scalar to quantify how costly a weight vector is.
      2. Need a norm of the weight vector: lambda times the summation of w\_k^{2} = lambda w^{T} w.
   4. As before, this corresponds to zero mean Gaussian weight prior.
      1. E(w) = negative product sum of {(yi times log p(y=1 given x\_i) + (1 – y\_i)log(1 - p(y=1 given x\_i))} + lambda w^{T} w.

* 1. Take derivatives of…
     1. The Posterior rather than the Likelihood
     2. The augmented cost function  
          
        w' = w + alpha(E(w) / dw)   
        w' = w – alpha times the summation of (sigma(w^T x) – y\_i) x\_i) – lambda w.

1. **Summary**:
   1. Logistic Regression
      1. Uses sigmoid function to predict probability in [0,1]
      2. Weights specify the decision boundary
      3. Weight magnitudes specify its sharpness
   2. Cost as negative log probability of correct assignment.
      1. Using binary exponent trick.
   3. Differentiation w.r.t weights leads to convex gradient-based solution.
   4. Regularize to avoid overconfidence
2. **Multiclass: 1 vs All**:
   1. Simple solution: 1-versus-All
      1. Use K classifiers, each solving a two-class problem of separating class k from all others.
   2. Issues:
      1. Ambiguous regions
      2. Gets expensive with many classes
3. **Multiclass: Multinomial logistic regression, MaxEnt, softmax**:
   1. Bernoulli likelihood: p(y=1 given x) = 1 / (1 + exp^{-w^T x}) = Bernoulli(y given sigma(w^{T} x))
   2. What was the “dice” rather than “coin” distribution?
      1. Multinomial. A parameter vector that adds up to 1.
      2. Now y=1…K, rather than y=0,1.  
           
         p(y given x, W) = exp(w^T\_y x) / sum of exp(w^T\_y x)  
           
         p(y given x, W) = multi(y given softmax(w^T\_y x))
4. **Softmax**:
   1. What’s the range of exp(w^T x) for x -> -inf or +inf?
   2. What’s the range of: exp(w^T\_y x) / sum of exp(w^T\_y x)
   3. What if x = kw\_y for one y, and x = -kw\_y’ for the others y’ != y?
      1. And if k -> inf?
5. **Relating softmax to logistic**:
   1. For Binary case K = 2.
      1. What happens to Softmax?  
           
         p(y=1 given x, W) = exp(w^T x) / (exp(w^T\_2 x) + exp(w^T\_1 x))  
           
         Simplifying:  
         = 1 / (1 + (exp(w^T\_2 x) / exp(w^T\_1 x)))  
         = 1 / (1 + (exp(w^T\_2 x) - exp(w^T\_1 x)))  
         = 1 / (1 + (exp(w\_1) - exp(w\_2)^T x))  
           
         Same as logistic before:  
         p(y=1 given x)) = 1 / (1 + exp^{-w^T x}))
6. **Evaluating a Multiclass Classifier**:
   1. Quantify The Goodness of a MaxEnt Classifier?
      1. Again, probability that all data assigned correctly.  
           
         p(Y given X, W) = double product sum of p(c given x\_i, W)^{Y\_k}
      2. Where we let Yic be a sparse binary 1-of-N matrix encoding the labels. i.e, If y\_i=c, then Y(i, c)=1.
      3. Dimensions of Y and W?
         1. Y is a N x K matrix of labels.
         2. Now W is a KxD matrix stacking D dims of weights for each of K classifiers w\_y’ .
      4. To specify a cost, take logs as usual:  
           
         E(W) = double sum of Y\_{ic} times log p(c given x\_i, W)  
         p(c given x, W) = exp(w^T x) / the sum of (exp(w^T\_2 x) + exp(w^T\_1 x))  
         E(W) = sum[ (sum of Y\_{ic} w^T\_c x\_i) – log(sum of exp(w^T\_c’ x\_i) ]
7. **Learning MaxEnt**:
   1. Differentiate the cost wrt W:  
        
      E(W) = sum[ (sum of Y\_{ic} w^T\_c x\_i) – log(sum of exp(w^T\_c’ x\_i) ]  
      dE(W) / dw\_c = sum of (p(c given x, W) – Y\_{ic}) x\_i  
        
      Intuition: If x\_i supposed to be class c:
      1. Push w\_c towards/away from x as appropriate
8. **Summary**:
   1. Multi-class classification:
      1. Simple indirect solution 1-vs-all of Logistic regressions
      2. Direct solution: Use softmax.
   2. Cost function: Probability of correct assignment.
      1. Use binary exponentiation trick again to express succinctly.
   3. Can learn with gradient as before.
   4. Same form as LR in the case where K=2.
9. **Non-linear trick**:
   1. As with Linear Regression, we can use a fixed non-linear (e.g., polynomial) transformation.
      1. Result: Non-linear in feature-space. Linear w.r.t w  
           
         p(y=1 given x)) = phi(w^T x) = 1 / (1 + exp^{-w^T phi(x)}))  
         where,   
         phi(x) = [1, x\_1, x\_2, x^2\_1, x^2\_1, x^2\_2, x\_1, x\_1]^T  
         Therefore,p(y=1 given x)) = sigma(w\_0 + w\_1 x\_1 + w\_2 x\_2 + w\_3 x\_2 + w\_5 x^2\_1 + w\_5 x\_1 x\_2)
   2. Using non-linear logistic regression this way, can result to over/underfitting again.
10. **Regularization for Sparsity**:
    1. Common to use l2 regularization:  
         
       E(w) = negative product sum of {(yi times log p(y=1 given x\_i) + (1 – y\_i)log(1 - p(y=1 given x\_i))} + lambda w^{T} w.
    2. As for linear regression, L1 is also possible:  
         
       E(w) = negative product sum of {(yi times log p(y=1 given x\_i) + (1 – y\_i)log(1 - p(y=1 given x\_i))} + lambda times the absolute values of sum w^{d}.
    3. This increases sparsity.
11. Sometimes we want sparsity.
    1. Delete irrelevant distractor features.
    2. Save memory.
    3. Gain domain insight.
12. How? Remember those regularizers…
    1. L0 is explicit about sparsity. Cost = # non-zero elements.
13. **L1 Regularization**:
    1. L1 regularization increases sparsity: L1 cost gradient remains constant approaching zero L2 cost gradient approaches zero as w approaches zero.  
         
       L1 represents a downwards arrow-tip function while L2 is more rounded like a bullet.  
         
       But L1 regularisation still harder to Optimize. Why?
       1. Absolute value has no derivative at zero (non-smooth corner).
14. **Multi-label classification:**
    1. Logistic Regression assumed data like {x\_i, y\_i}.
       1. And trained predictor:  
          p(y=1 given x) = sigma(w^T x) = 1 / (1 + exp^{-w^T x})
    2. What about if you have {x\_i, y\_i}? (Vector of labels)
       1. Classification analogy of multiple output linear regression.
       2. When?
          1. E.g., Webpage topic classifier for indexing. One web page can have multiple topics: Education+Government, Business+Finance, Business+Movie Industry..
          2. Face photo has multiple attributes: Age, Eyeglasses, gender, long hair, hair colour, mustache, …
          3. Movie/music can span multiple genres.
15. **Multilabel vs Multiclass**:
    1. Multiclass
       1. One x has exactly one label
       2. Train data is scalar y for one x
       3. Prediction y=f(x) is a scalar
       4. y=1….K exclusive
       5. P(y given x) is a K-sized vector
    2. Multilabel
       1. One x has many labels (0 to K)
       2. Train data is vector y for one x.
       3. Prediction y=f(x) is a K-vector
       4. Y\_k={0,1}
       5. P(y given x) is a K-sized vector
16. **Multilabel classification:**
    1. Given data {y\_i ,x\_i}.
       1. Train an independent logistic regression classifier for each pk (y\_k given x). – Stack up the results p(y given x) = [p1 (y1 given x)…..p\_K (y\_K given x)].
       2. This does not take into account label dependencies.
17. **Large Scale**:
    1. As for linear regression, the strategy is to minimise the cost function.

E(w) = negative product sum of {(yi times log p(y=1 given x\_i) + (1 – y\_i)log(1 - p(y=1 given x\_i))} + lambda times the absolute values of sum w^{d}.

* 1. By iterating:
     1. w^{s+1} = w^{s} – alpha times the sum of (sigma(w^T x) – y\_i) x\_i) – lambda w
  2. Means that you have to read database off disk into memory very many times.
     1. – Notice inner loop over i=1…N
        1. … within an iterative loop over s
  3. Regular:
     1. Iterate S times:  
          
        w^{s+1} = w^{s} – alpha times the sum of (sigma(w^T x) – y\_i) x\_i) – lambda w
     2. SGD with mini-batches:
        1. Iterate M times:
        2. Select a random subset B of total N examples
        3. Iterate S times.
     3. Before: S memory reads of full database. – Now M\*(B/N) reads. Can be << S. – E.g., M=N/B => Read once.

1. **Interpreting Logistic Regression Outputs**:
   1. Once learned:
      1. Recall each w\_k is multiplied with x\_k in sigma(w^T x)
      2. Positive w\_k => Feature is indicator of class.
      3. Negative wk => Contra-indicator.
      4. If L1 regularizer killed w\_k
         1. Irrelevant to classification.
2. **Summary**:
   1. Non-linear extensions exist
      1. Like linear regression, take fixed polynomial basis functions.
         1. Then you would need to regularize for more than ‘just’ overconfidence.
   2. L2/L1 regularizer options also exist.
      1. L2 is efficient.
      2. L1 less efficient but helps find sparsity.
   3. Multi-label versus Multi-class classification.
      1. Multi-label: Simple solution by K-binary classifiers.