**Machine Learning**

**Session 3**

1. **Linear Regression (LR)**:
   1. Predict the value of one or more continuous target *y* variables given a *D*-dimensional vector *x* of inputs.
   2. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model.
2. **LR Settings:**
   1. *One-dimensional linear*:
      1. Given a training set of a variable *x* ranging from *x\_1* to *x\_n*, and a variable *y* ranging from *y\_1* to *y\_n*.
      2. Learn weights (decides how much influence the input will have on the output).
      3. Predict new variable value given the other.
      4. Represented as a straight-line graph with data points around the line-of-best-fit (LOBF).
   2. *N-dimensional linear*:
      1. As the number of features grows, a linear plane intersecting the data points on a 3D-plot is used to draw relationships between multiple variables.
      2. Represented as a plane in 2D space. Linear w.r.t to both variables.
   3. *N-dimensional polynomial non-linear*:
      1. Represented as a sine wave with data points along the wave. Linear w.r.t to only one variable.
3. To **find the optimum weight** between different settings:
   1. *Error (Cost) function*:
      1. Quantifies the error between predicted values and expected values and presents it in the form of a single real number i.e., accounts for the distance of the data points from the LOBF in a 2D graph.
      2. Most common cost; sum of squared errors:
         1. E(w) = summation of yi minus the function representing the straight-line produced by xi times w; all squared.
      3. Find the best LOBF by choosing weights such that cost/error is as small as possible.
      4. Goal is to find the line (or hyperplane) that minimizes the vertical offsets. Or, in other words, the LOBF is defined as the line that minimizes the sum of squared errors (SSE) or mean squared error (MSE) [convex functions; more below] between the target variable *y* and the predicted output over all samples *i* in the dataset of size *n*:
         1. *Gradient*: Take the derivative of the function w.r.t the weights *w*. This gives the gradient at any point on the graph. To minimise, move towards the lowest values until converged or the gradient is negligible.   
              
            If a cost function is represented as a wave on a graph with different peaks and troughs, to ensure that converge is global rather local i.e., the function minimises to the lowest trough on the whole graph space instead of the lowest trough in the immediate graph space, a *convex* cost function is used.   
              
            A function *f* is said to be a convex function if the seconder-order derivative of that function is greater than or equal to 0 i.e., positive.  
              
            Convex functions only have a single global minimum. Learning rate (alpha) must be set low to avoid divergence and overshooting.
         2. *Closed-form solution*: Exact, non-iterative solution. But in-memory requirement and matrix inversion mean only small datasets can be used.
4. **Nonlinear regression**:
   1. Nonlinear regression is a form of regression analysis in which data is fit to a model and then expressed as a mathematical function. Simple linear regression relates two variables (*X* and *Y*) with a straight line (*y = mx + b*), while nonlinear regression relates the two variables in a nonlinear (curved) relationship.
   2. The goal of the model is to make the sum of the squares as small as possible. The sum of squares is a measure that tracks how far the *Y* observations vary from the nonlinear (curved) function that is used to predict *Y*.
   3. It is computed by first finding the difference between the fitted (using basis function: standard sine/cosine wave to fit any given curve) nonlinear function and every *Y* point of data in the set. Then, each of those differences is squared. Lastly, all the squared figures are added together. The smaller the sum of these squared figures, the better the function fits the data points in the set.
5. **Underfitting**: Occurs when the model over-generalizes and fails to incorporate relevant variations in your data that would give your model more predictive power. You can tell a model is underfitting when it performs poorly on both training and test sets.
6. **Overfitting**:
   1. Occurs when the model learns the training data too well and incorporates details and noise specific to your dataset. You can tell a model is overfitting when it performs great on your training/validation set, but poorly on your test set (or new real-world data).
      1. Tuning or selecting the preferred level of model complexity so the models are better at predicting (generalizing).
      2. Add a penalty to the cost function to discourage heavy use of all the weights.
      3. *Validation set*: Split data into training and validation sets. Monitor the performance of the model on the validation set after training.
7. **Regularizers**: Regularisation parameter (lambda) controls the complexity… and hence degree of over/under fitting by defining a trade-off between fit and weight decay.  
     
   Allow the application of penalties on layer parameters or layer activity during optimization. These penalties are summed into the loss function that the network optimizes.  
     
   Learning with regularization. – Usual procedure:
   * 1. Differentiate E(w) w.r.t w and get gradient or closed-form solution.
   1. Determining Regularisation Strength:
      1. Use a validation set – Split your data X into:
         1. Xtr Training set: determine w
         2. Xval Validation set: tune lambda
      2. Performance on Xval reflects performance on (the unknown) Xtest.
      3. Because the data for training w excludes Xval:
         1. If w is overfit, it will perform badly on Xval
         2. “Safely” use performance on Xval to determine the right complexity.
      4. Pay attention to the data split in training/validation. Usually 60%/40%.
8. **Probabilistic Regression**:
   1. *Likelihood function*: See PDF in the same folder. His notes are a mess.