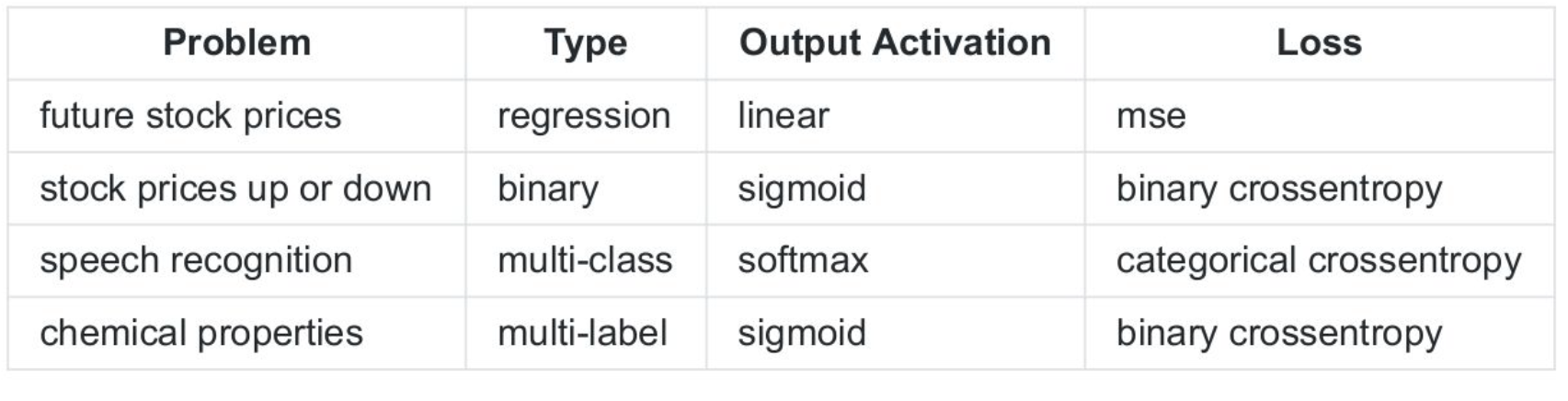
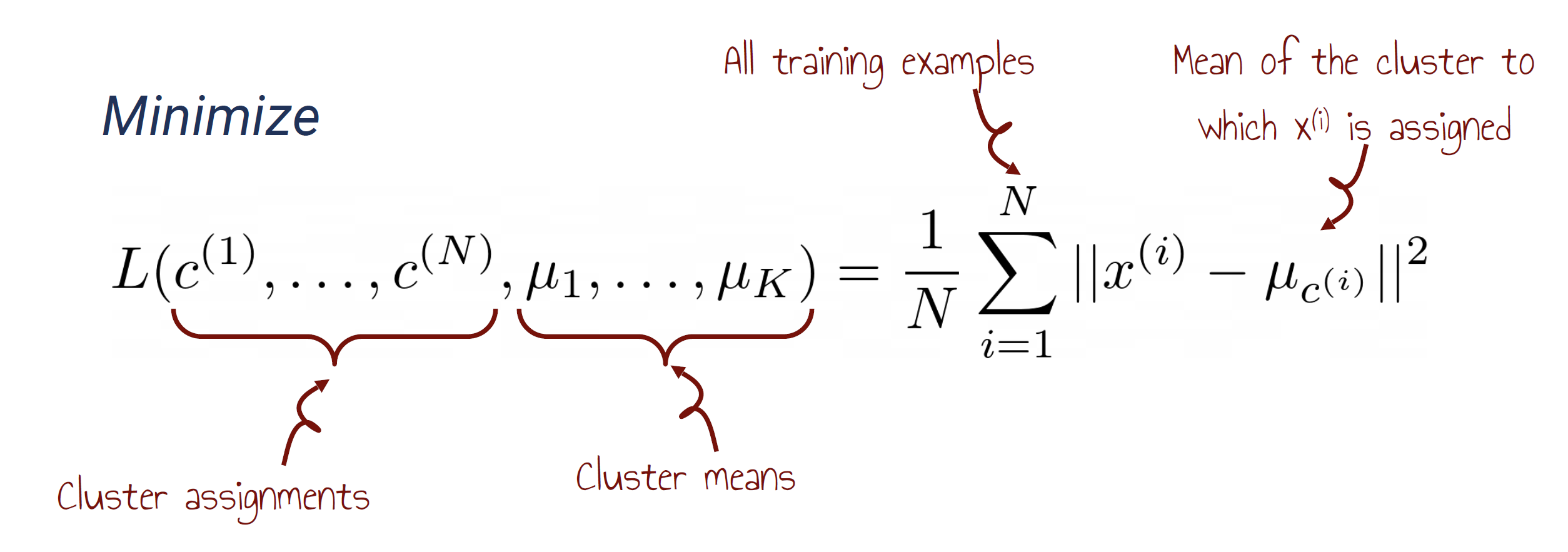
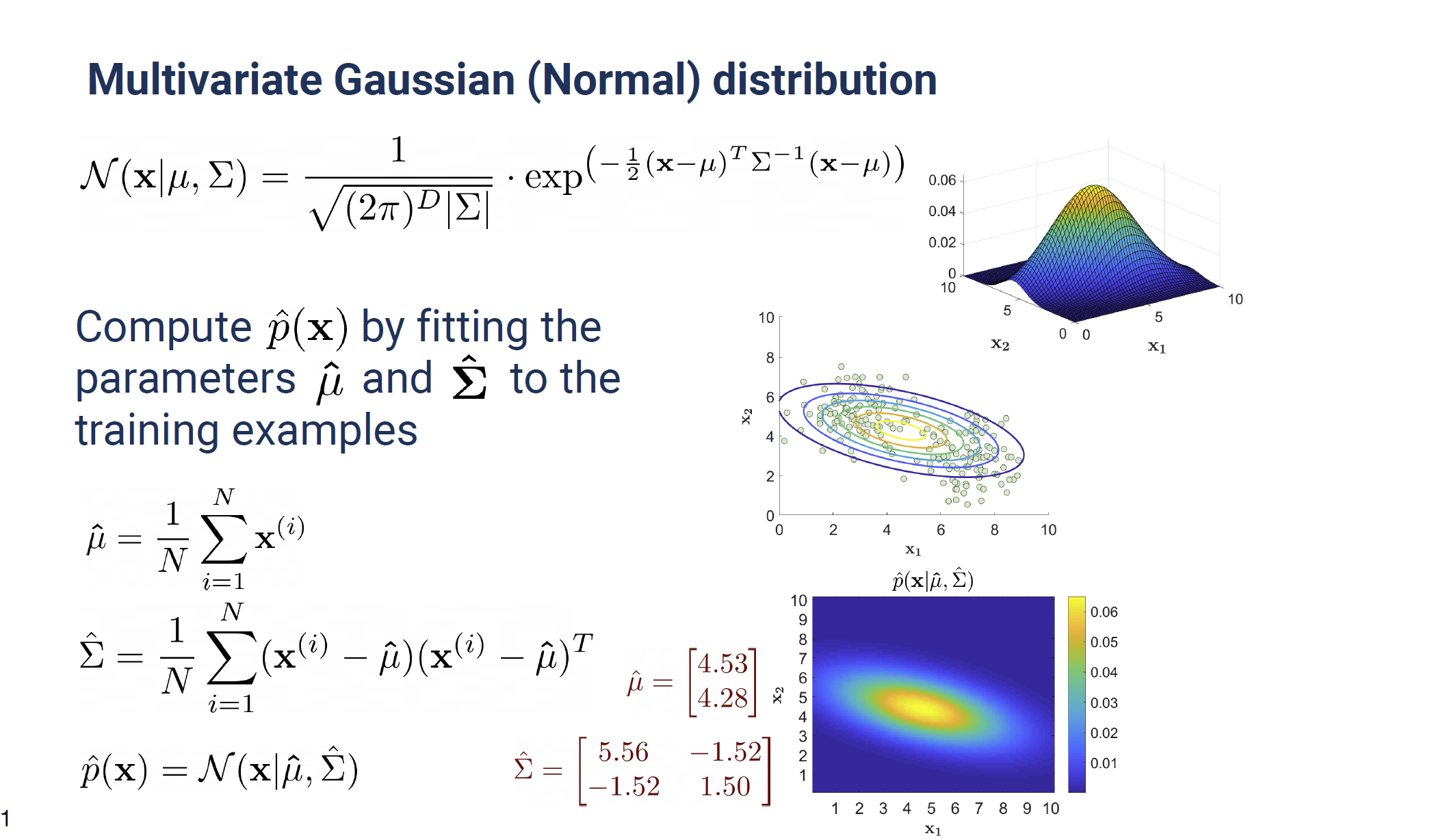
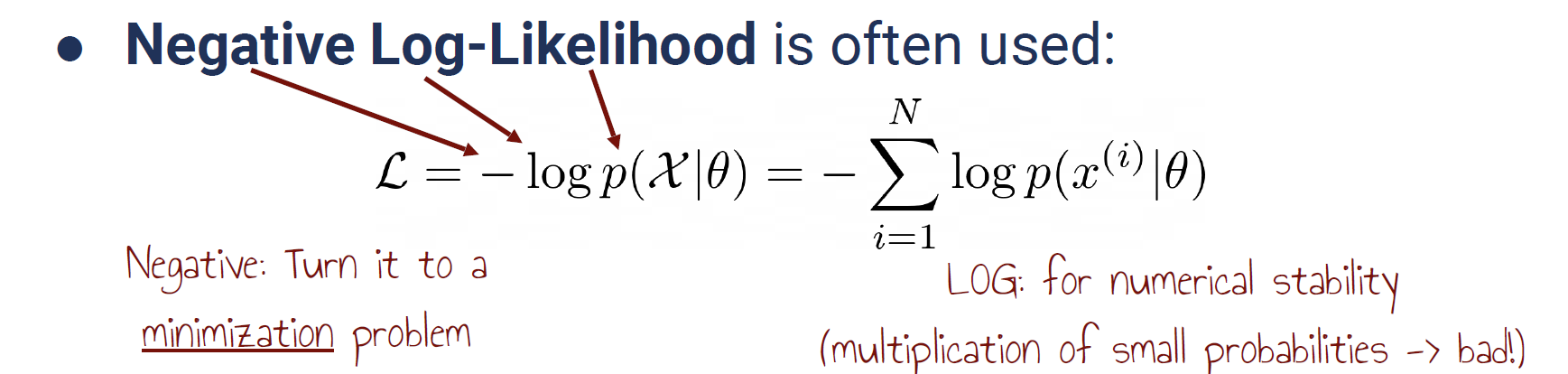
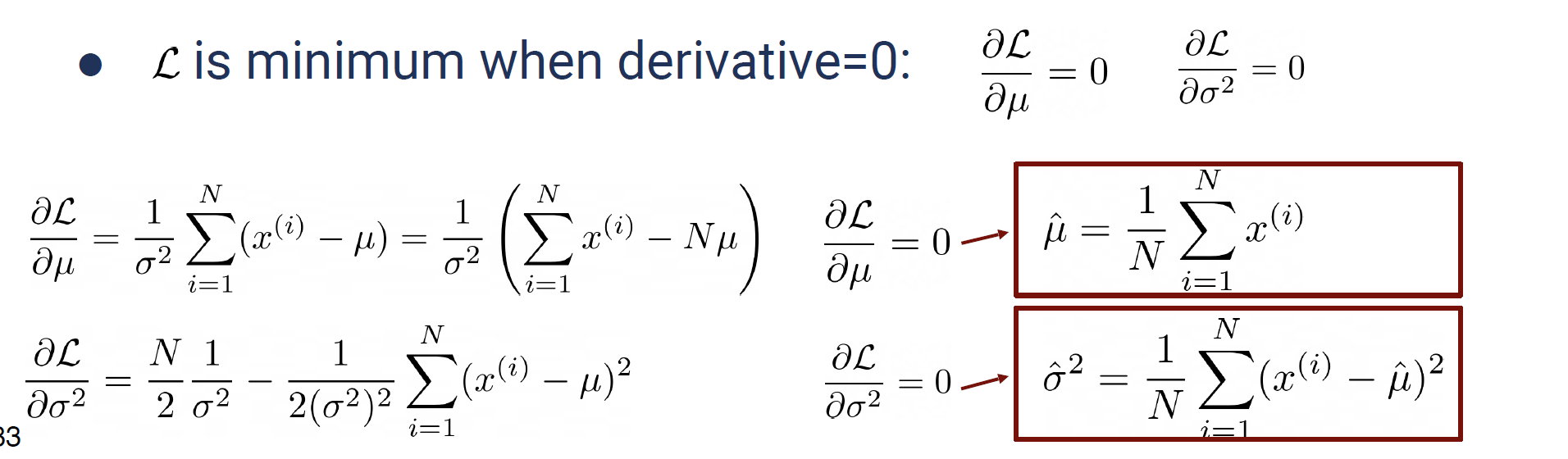
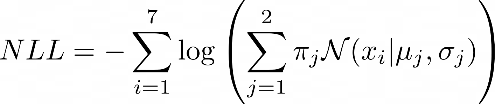
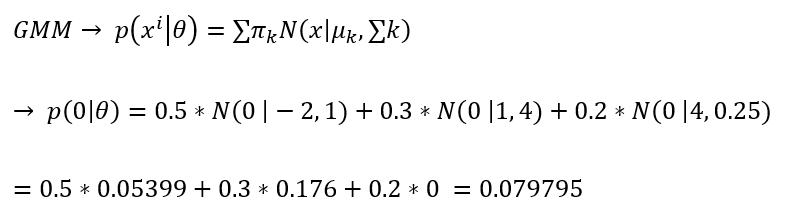
**Instanced-based Learning & Decision Trees**

1. **Lazy learner:** K-Nearest Neighbours
2. **Eager Learner:** Decision Tree
3. **Nearest Neighbor Classifier:** KNN is conceptually simple and has the advantage of being nonparametric. That is, the method can be used even when the variables are categorical—though if you are using numeric variables in the mix, it is best to standardize them
   1. **Increasing K:** Underfitting. Having a smoother decision boundary (higher bias) and less sensitive to training data (lower variance)
   2. **Distance Metrics:**
      1. For a given problem with a fixed (high) value of the dimensionality d, it may be preferable to use lower values of p. This means that the L1 distance metric (**Manhattan Distance** metric) is the most preferable for high dimensional applications.
   3. **Weighted K-NN:** weight of the neighbor is assigned with a weight that equals to the inverse of distance. The value of k is of **minor importance**  because distant examples will have small weights and won’t greatly affect classification. This is robust to noisy training data
   4. **K-NN Curse of Dimensionality:** Increasing the number of features will cause instances that should belong to the same class are actually separated far from each other in certain dimensions.
4. **Decision Tree**
   1. A skewed distribution has a **low** entropy, whereas a distribution where events have equal probability has a **larger** entropy. Essentially, with a larger entropy, we have less knowledge.
   2. The goal for decision tree in each iteration is to select an attribute / split method that can maximize the information gain defined by (Entropy\_before – Entropy\_after)
   3. **Continuous Entropy** with integration form is also adopted for probability density function instead of discrete variables
   4. **Different Inputs**
      1. **Ordered Values:** For each feature, sort its value and consider only split points that are between two examples with different class labels
      2. **Categorical Values:** Search for the most informative feature and then create as many branches as there are different values for this feature
5. **Random Forests:** Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees.
6. **Regression Trees:** Use training examples at leaf node to estimate the output value (e.g., value that minimize min squared error) or learn some linear function of some subset of the numerical features

**Evaluation**

1. 
2. **Hyperparameter Tuning:** Try different hyperparameter values on the training dataset, then select the best according to the accuracy on the validation dataset. Perform the final evaluation on the test dataset.
3. NEED TO SHUFFLE THE DATASET FIRST BEFORE SPLITTING!!!!
4. **UnderFitting:** The neural net seems to be under-fitting: we expect models to be able to reach error rates of around 10%, but this one has about 40%. It does not seem able to learn more than this, thus, we should change its structure (number of hidden layers, number of neurons per layer, activation function) in order for it to become more complex and become able to learn more and fit our data better. This can also be due to the learning rate being so small that the optimization gets stuck in a local minimum, so increasing the learning rate might help. In the case of it underfitting, introducing learning rate decay can also help, as it could be the case that the network is “bouncing around” in a bowl containing the global minimum but can’t reach it if we don’t decay the learning rate introducing. There might also not be enough training data.
5. **Cross-Validation:** Divide dataset into k equal folds/splits. Use k-1 folds for training + validation and one for testing. **Global Error Estimate =** average value of all observed error
   1. At each iteration, use 1 fold for testing, 1 fold for validation and k-2 folds for training
   2. At each cross-validation step separate 1 fold for testing. Run an internal cross-validation over the remaining k-1 folds to find the optimal hyperparameters.
      1. 10k datapoints - enough data, can do 80/10/10 split
      2. Shuffle the data
      3. K fold validation
      4. Iteratively leave 1 fold for testing
      5. Training + validation of k-1 folds
      6. Hyperparameter tuning
      7. Repeat for all validations
      8. Aggregate & average accuracy
      9. Get the best values on the left out fold each and get the best params for each split
6. **Before PRODUCTION,** we can use all available data for training the model with found best hyperparamter (i.e., K value in K-NN model)
7. **Recall & Precision**
   1. Recall means the number of positive examples that are correctly recognized. Precision means the number of positive predictions that are actually positive
   2. **High Recall Low Precision:** Most of positive examples are correctly recognized but there are many false positives
   3. **Low Recall High Precision:** Miss lots of positive examples but those we predict as positive are really positive
   4. **Macro-Averaged Recall:** Recall = TP / (TP + FN) or TN / (TN + FP). Compute the mean between these two measures
   5. **F-Measure:** Beta means ratio of importance between recall and precision. **If Beta = 2, recall is twice important than precision**
8. **Imbalanced Dataset:** With a imbalanced dataset, the macro-averaged recall & Precision will be severely affected (i.e., Class-based Measure). Macro-averaged recall can be helped detecting if one class is completely misclassified, yet no information about the FP.
   1. F1 is most suitable, as it takes into account both precision and recall for each class.
   2. Accuracy is biased towards the majority class, therefore not suitable for imbalanced data.
   3. To mitigate, can oversample minority classes to balance out the distribution.
   4. **Solution:** Normalize Matrix (i.e., assume if we had a balanced number of examples)
   5. **Downsample / Upsample.** This will help balancing the classes in training dataset, but it will not reflect the ability of model to generalize.
9. **Confidence Interval**
   1. Error\_h(S) = 1 – accuracy of a particular class.
10. **Statistical Significance**
    1. The statistical significance tries prove if two algorithms/models perform the same and the performance differences are only due to sampling error. With a **smaller p-value**, we can confident that one system is indeed different from another.
    2. P<0.05 means statistically significant. P>0.5 **does not prove** two algorithms are similar but only prove we cannot observe significance.
    3. **It is important to note that when a statistical test fails, this does NOT mean that the two algorithms are comparing equally. Instead, it means that we cannot statistically rule out the probability that these two algorithms are the same or different. In short: we cannot conclude.**
    4. **So, here what we want to do is to be able to conclude. One approach could be to run more training sequences to increase the strength of the analysis. Ideally, we should also run a "power-analysis" to evaluate if the number of training sequences is enough.**
11. **Early Stopping**
    1. There is a strong assumption going on behind the scenes of this intuitive principle. It assumes that validation accuracy/loss should monotonically increase as we train, and if it doesn’t, we are overfitting our training set. A less obvious but equally important assumption is that there is a strong correlation between loss and accuracy.

**Unsupervised Learning**

1. **Clustering:** Grouping instances together such that instances in the same group are more similar to each other than to other instances in other groups
   1. **Vector Quantization:** It works by dividing a large set of points (vectors) into groups having approximately the same number of points closest to them. Each group is represented by its centroid point, as in k-means and some other clustering algorithms.
   2. **Image Segmentation:** It creates a pixel-wise mask for the objects in an image which helps models to understand the shape of objects and their position in the image at a more granular level.
2. **K-Means Algorithm**
   1. **Stages & Overview**
      1. Initialization => Assignment => Update
      2. **Initialization** is better to select K instances at random to avoid empty clusters
      3. **Assignment** calculates for each training datapoint, to which cluster does this training example x(i) has the minimum distance to the cluster mean (centroid) by calculating with (x1 – mean) \* (x2 – mean)
      4. **Update** calculates the new cluster mean with all newly assigned training datapoints
      5. **Converge** if the difference between the updated mean and previous mean is insignificant enough (i.e., e)
      6. Varonoi diagram is drawn with equally-distanced from two adjacent means
      7. 
   2. **Hyper-Parameter Tuning & Evaluation**
      1. **Elbow Method:** Select K where the rate of decrease sharply shifts
         1. In order to determine the optimal number of clusters for the points, we could run the algorithm with several values of k, plot the score for each value of k, which should yield a decreasing convex curve. By using the elbow method, we could look at the point of inflection of the curve, which is the point where the model fits the data the best without being too complex.
      2. **Cross-validation:** For each iteration (with 10-fold cross validation), we compute the loss function in validation set with k-means of values from 1…10 and select a K such that further increase in number of clusters leads to only a small improvement in the average score k. **This can get preview on model’s performance on unseen data.**
      3. **Strength:** Simple, Popular, Efficient with O(TKN) for T iterations, K clusters and N training examples
      4. **Disadvantages:** K-means only finds local optimum, and sensitive to initial centroid positions. Sensitive to outliers, and only applicable when a distance function exists, not suitable for clusters that are not hyper-ellipsoids
      5. **k-means can be viewed as a special case of GMMs (more precisely, a limit case where the variances are taken to zero).**
3. **Probability Density Estimation**
   1. **Density estimation** is the construction of an estimate, based on observed data, of an unobservable underlying probability density function. The unobservable density function is thought of as the density according to which a large population is distributed; the data are usually thought of as a random sample from that population.
   2. **Non-Parametric Approach**
      1. No assumption on the form, and the number of parameters grow with data. **Expected for High Variance and Low Bias.**
      2. **Histogram:** Less bins => Smoother PDF generated
      3. **Kernel Density Estimation:** Assume Kernel Function H as a Gaussian (Smoother PDF generated)
         1. **Bandwidth Selection:** A **small** bandwidth leads to undersmoothing. It means that the density plot will look like a combination of individual peeks (one peek per each sample element). A **huge** bandwidth leads to oversmoothing. It means that the density plot will look like a unimodal distribution and hide all non-unimodal distribution properties (e.g., if a distribution is multimodal, we will not see it in the plot).
   3. **Parametric Approach**
      1. Simplified assumption about the form with fixed number of parameters. Low variance but introduces higher bias.
      2. 
      3. **Use Likelihood to evaluate the performance of model.** This measures the probability of observing data x from the dataset.
      4. 
      5. **It is noted that in order to optimize the log-likelihood, it is same as using the mean of all data points & variance of data points.** 
      6. **TO Compare two GMM-EM Model, use **
4. **GMM-EM Model**
   1. **The reason why we cannot adopt likelihood maximization is because** the function l(θ|Sn) is unbounded, so the true maximum is +∞ and corresponds to μ^(i)=x1 (for instance) and σ^i=0. A true maximiser should therefore end up with this solution, which is not useful for estimation purposes.
   2. Even without considering the kn terms in the decomposition of the product of sums as a sum of products in l(θ|Sn), the function to be maximized in θ is highly multi-modal (in addition to being non-convex) hence a challenge for numerical methods. EM acknowledges the difficulty by converging to a local mode or saddle point and requiring multiple runs.
   3. 

**Evolutional Algorithms**

1. **Motivation:** It means that the objective function is not a nice closed form function. But still we are interested in optimizing this function and get the values of the variables. In this case, as the function is unknown, we cannot compute gradients. But still we need to optimize. So we go for numerical methods of optimization or population based approaches that need only the value of the fitness function, which in this case is your black box (objective) function.
2. **Terminology**
   1. **Gene:** Sequence of nucleotides in DNA that codes a particular trait
   2. **Genotype:** Set of genes. Contains information that coded and stored in DNA
   3. **Phenotype:** The objective function based on parameters (i.e., Genotype)
3. **Fitness Function:** Maximization problem as we are looking for the fittest individuals.
4. **Behavior Descriptor:** It is used to define different types of solution and use to **compute distances between solutions**

**Useful Genetic Algorithm Template for Examination (18-19 Exam 3)**

* 1. So with three motors, we need to find the values of 3 parameters.
  2. **Genotype:** As each motor can take up to 11 possible discrete values, we can use a binary string with 4 \* 3 bits.
  3. **Phenotype:** We aggregate the bits 4 by 4, and each 4 of bits is mapped to a discrete possible angular position that motor can take.
  4. **Development of Genotype to Phenotype:** For each consecutive four bits, they are mapped to a possible angular position.
  5. **Selection Operator:** Uses tournament selection operator by firstly randomly drawing two individuals from the population, and select the best out of two. Repeat the entire process until we have N/2 pairs of parents.
  6. **Cross-Over Operator:** Adopts Single-Point Cross-Over by selecting a random split point among two parents, and form the offspring by exchanging the portions of genotype in the parent. **NEED TO FIND SPLIT POINTS BETWEEN TWO GENES**
  7. **Mutation Operator:** For each bit of the genotype, a number is randomly generated between 0 and 1, and if the number if lower than 1/12, then the bit is flipped.
  8. **Fitness Function:** We can use the negative distance to target as through maximizing this metric achieves our desired goal (0 distance = touching target). **Potentially adding special cases for phenotypes that are invalid and forcing them to –inf so that they are discarded in next gen**
  9. **PsuedoCode**
     1. Randomly generate a population of N binary strings
     2. Evaluate the fitness of each individual
     3. Select N/2 couples of parents w.r.t. the fitness using the selection operator
     4. For each couple, apply cross-over operator to create offspring
     5. Apply mutation operator on each offspring
     6. Replace the old population with the new one, and loop from stage 2.

**Invalid Genotypes**

There are multiple ways to deal with invalid genotypes of the encoding.

You can do it in the fitness function: you can assign a very bad fitness value (e.g., the lower bound if the fitness function is bounded). This is the most common approach as it is quite simple to implement.

You can change the mutation operators so that if they apply a mutation that makes the solutions not viable then the mutation is resampled until it works.

You can probably do the same with the cross over operator.

**Justification for Using Hidden Layers**

For the justifications: 1) the problem is not linearly separable, so we need at least one hidden layer, 2) we need at least 3 neurons (i.e., 3 line boundaries to be connected together) to encapsulate the black dots.

For the output neuron, it is better to use a different activation function, like tanh, so that you can obtain directly a probability.

**Activation Function:** Should be Differentiable & Non-linearity

