Chapter 1

- **Overfitting in linear regression is due to the presence of a large number of coefficients (use hypothesis testing to determine which to keep), a higher degree polynomial, thus leading to misleading R-values, however, with a higher degree polynomial errors are reduced as compared to linear least-squares fit. Computational complexity increases for a higher degree of polynomial functions.**
- LMS works for arbitrary functions, as long as the minimum can be found by gradient descent (error surface is convex).
- LMS can still be applied if the error function is non-convex. In this case, it may find only the local minima.

MAE or the L1 loss - Overshoots the error. **More robust to outliers,** but its **derivatives are not continuous** making it inefficient to find the solution. It is the sum of absolute differences between the target and the predicted variables. So, it measures the average magnitude of errors in a set of predictors, without considering directions (if directions are considered that is called **Mean bias error (MBE);** range (0, inf]).

Linear model complexity is 1.

MSE (*captures similarities in data, captures both bias and variance*) or the L2 loss or Quadratic loss are **sensitive to outliers,** in this case, use any other loss function (hinge loss or quantile loss). Also, gives a more stable and closed-form of the solution by setting its derivative to 0. **MSE answers: How accurately have we estimated the parameters of the data?**

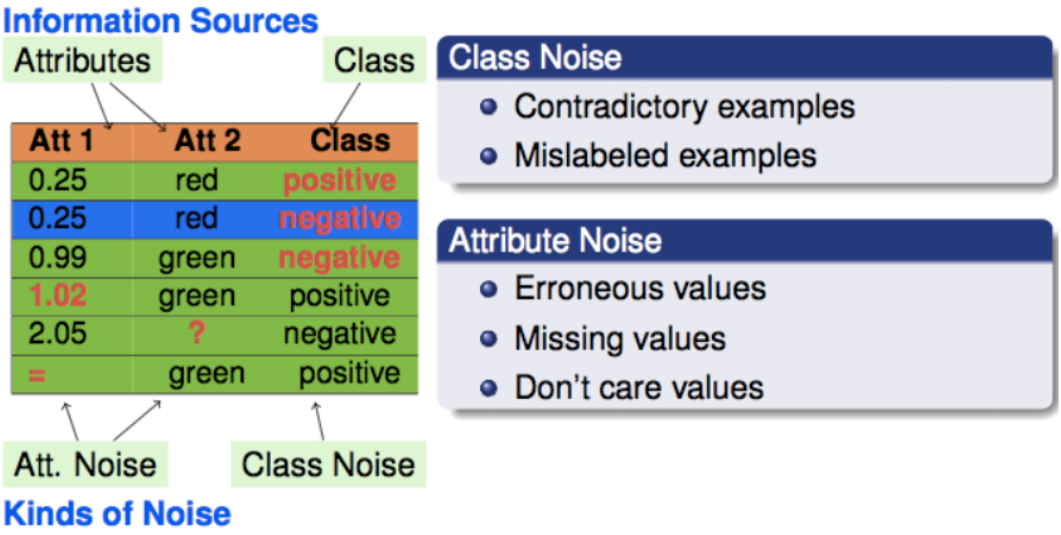
Outlier- That does not belong to the range of your data points.

Which loss function to use? "If the outliers represent anomalies that are important for business and should be detected, then we should use MSE. On the other hand, if we believe that the outliers just represent the corrupted data, then we should choose MAE as loss".

Quantile loss: For modeling, the uncertainty in our models gives the **range of predictions** as opposed to 'point estimates' and can significantly improve decision-making processes for many business problems.

Causes for outliers in data:

1. Data entry or measurement errors,
2. Sampling problems and unusual conditions.
3. Natural variation in the data distribution.



Performance metrics (Binary predictors)

- **Precision** is useful where **FP** is a higher concern than FN (**music or video recommendation systems where wrong results could lead to customer churn and be harmful to business.**)
- The **recall** is useful where **FN** is more important than FP (**In the medical domain, where it does not matter whether we raise a false alarm but the actual positive cases should not go undetected**).
- Accuracy is used when the **TP** and **TN** are more important (less important in medical). Made use when the **class distribution is similar.** Suffers from the so-called **'Accuracy Paradox'- it is sometimes desirable to select a model with a lower accuracy because it has greater predictive power on the problem. A dataset with a large class imbalance, highly biased dataset model therefore will predict the value of the majority class for all predictions and achieve very high accuracy.**
- F1 score is used when **FN** and **FP** are crucial. Used when there are imbalance classes as most real-life problems in classification have this. (Also alternatively can use Balanced accuracy, BER = (Sensitivity +

Specificity)/2; can be used for binary and multiclass problems). **Best value 1; worst value 0.**

- When the **predictors are continuous or ordinal scale** we use ROC (applications: medical diagnostic test). They represent the **performance averaged over all possible cost ratios**.
- If two ROC curves **do not intersect**, means one method dominates the other.
- If two ROC curves **intersect**, one method is better for some cost ratios, and the other is better for other ratios.
- The **learning curve** is a plot of model learning performance over experience or time (a line plot of learning (y-axis) vs. experience (x-axis)).
- Learning curves are a widely used diagnostic tool in ML for algorithms that learn continuously from a training dataset incrementally. Reviewing LC's of models during training can be used to diagnose problems such as underfitting or overfitting or whether training and validation sets are suitably representative.

Chapter 2

- The ID3 decision tree is **only used for classification** and not for regression. For regression trees use C4.5 or C5.0.
- **Classification -> labels/targets -> output is discrete.**
- **Regression -> quantity/magnitude -> output is continuous.**
- 0/1 loss function primarily used for the classification problem, but **can be also used for regression problems** too (ε-boundary).

Distributions

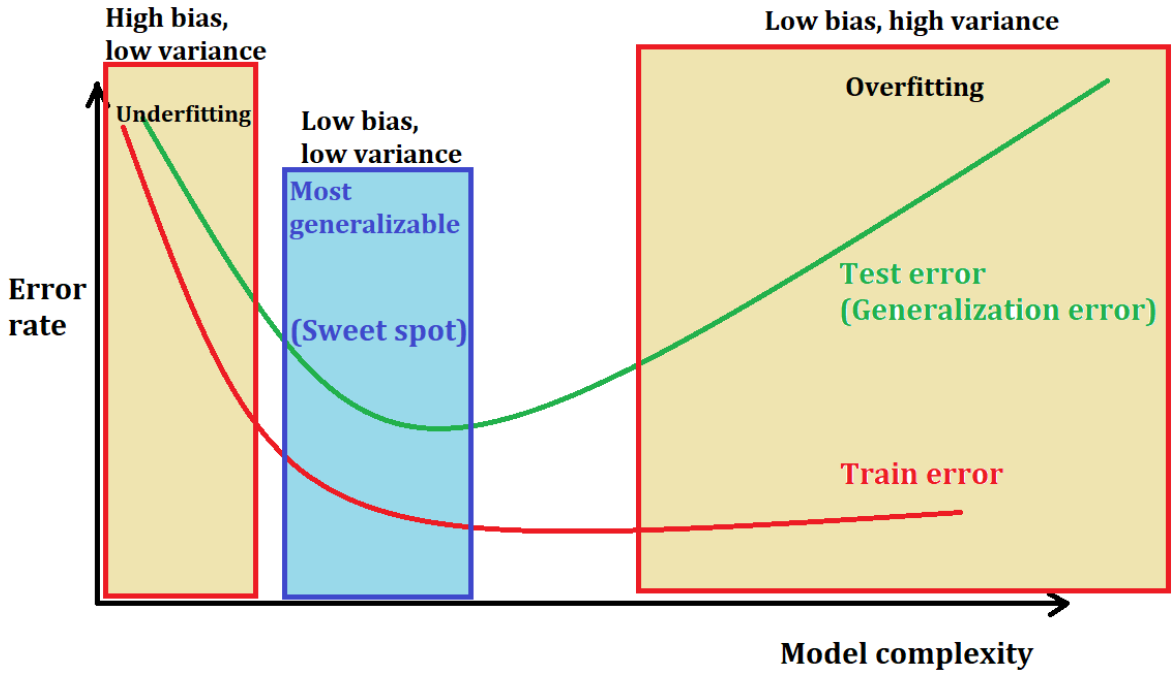
- Words in written text - multinomial distribution (categorical)
- Received emails/day or busy times (peak hours) in the supermarket - Poisson distribution (categorical)
- Material decay time - exponential distribution (numerical)
- Blood pressure levels - gaussian distribution (numerical)

Bias-variance tradeoff

Expected answer:
"Bias can be seen as the **number of assumptions** made in the model and therefore is to a degree the inverse to the model complexity. It can be estimated by the training error (irreducible error, linear models have many strong assumptions) *and draw a 5 cm 45 deg line on a sheet of paper without the help of a ruler*.

Variance is the difference of predictions between the test and training error. A large variance indicates that the model performs much differently on unseen data (possibility of reducing to a certain extent).

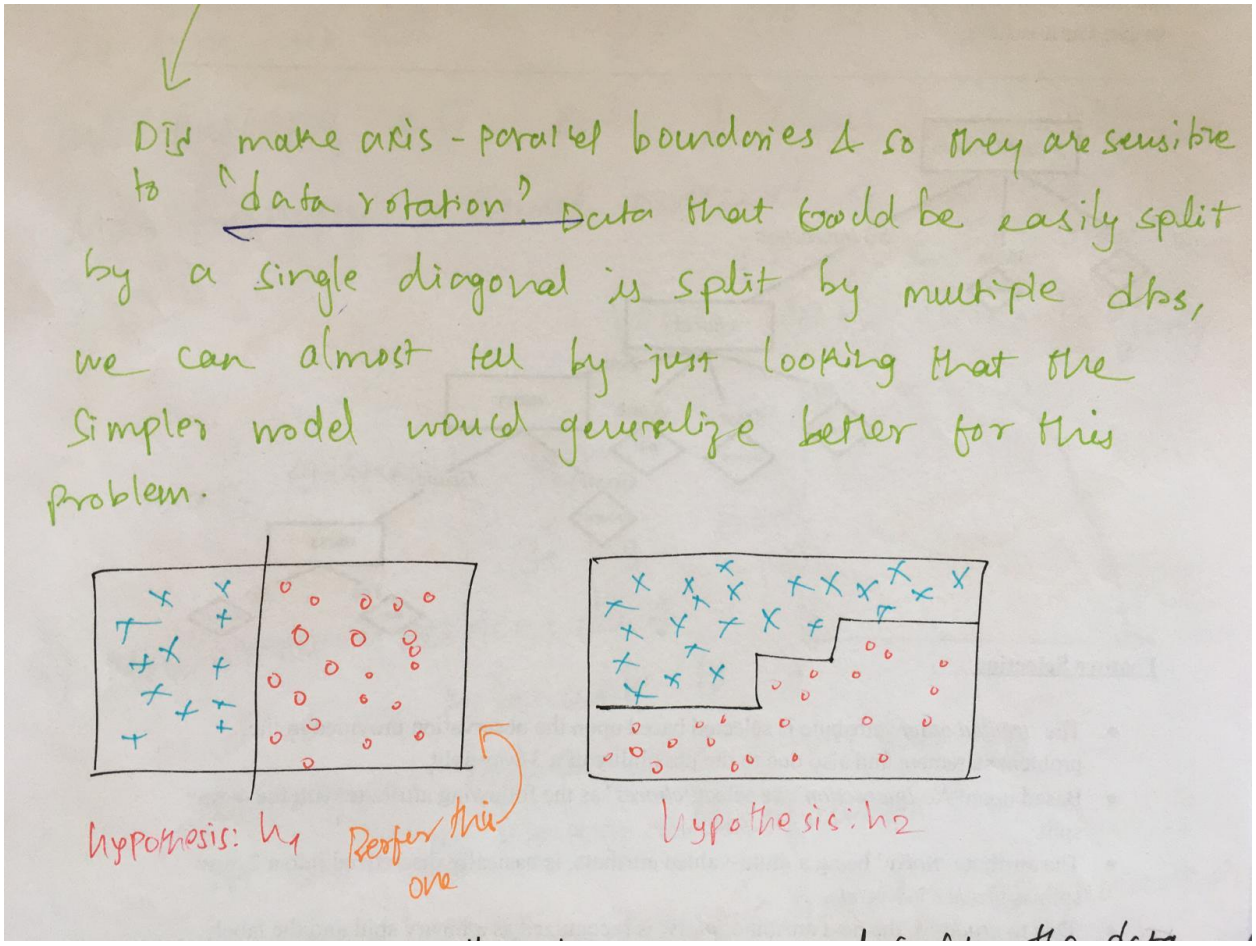
Therefore, a **low bias (high complexity model) and high variance (large difference in training and test error) usually indicate overfitting**".



Chapter 3

- Instances are **not only represented** only by the conjunction of constraints but can be modeled in other ways like squares or rectangles.--CE
- Hypothesis space (H) and version space wrt H, D is consistent if there are **no training errors**.
- Take **ONE** training example then check whether it accepts or rejects with the VS and based on this define the error.
- Hypothesis space is defined by the **user or the concept learner**.
- The find-S algorithm only considers **positive** examples and only works on the **conjunction of constraints and DOES NOT work on anything else**.
- Candidate elimination (sensitive to noise) **can hold ANY OTHER KIND** (ex. squares, rectangles) other than the conjunction of constraints.
- If training data contains **NO ERRORS then it will converge consistently. Missing values -> converges to empty version space.**
- **Requires noise-free training data**, if this condition is not met then the convergence will take longer.

- Boolean decision tree for concept learning -> every boolean function is evaluated from L->R
- Overfitting in DTs caused by **more nodes not proportional to class/target, and with more splits, data complexity of underlying distribution, model complexity**.
- Application of concept learning in DTs: Check which *h* works best (as an evaluation) than all the others which classify most test instances correctly. (*look figure below*)



K-fold Cross-validation (Rotation estimation)

- **Biased** towards k-folds, training error will be very different, **shuffle** operation must be performed once on the dataset. **Training errors will be different.**
- **Higher the k better is the model estimation and hence higher will be the variance.**

- Evaluation score by means of average (mean) => Is this average a good measure? Just taking the mean would be a "**Data murdering**" so wiser to consider the **variance** additionally of the model to know the difference or high from the mean.
- LOOCV is **exhaustive & unbiased,** as almost always the whole dataset, often very expensive to implement. **Training error will be potentially very similar.**

LOOCV	*k*-fold CV
A special case of *k*-fold CV when $k = N$	Dataset grouped into chunks of *k* sizes ($k<N$)
N-1 for training; 1 out of *N* for test	*k-1* fold for training; 1-fold for test
Computationally expensive to perform	Computationally economic to perform
Convergence time is higher (**use when the learning curve is steep**)	Better convergence in comparison to LOOCV

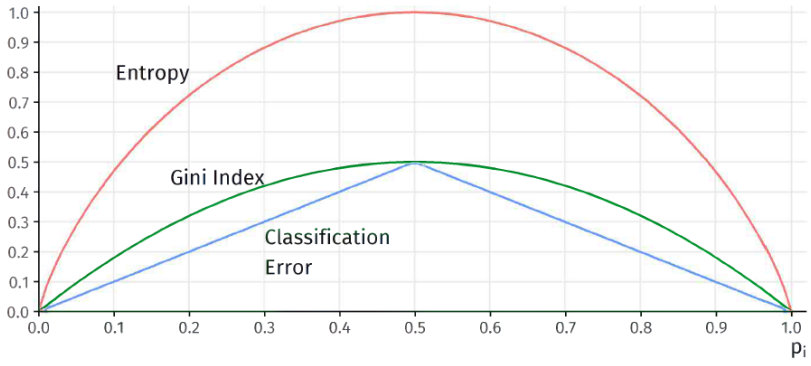
Chapter 4

Goal: recursively partition the training data into increasingly class-**purer** subsets

Impurity of a node t:

- **Entropy**: $H(t) = -\sum_{i=1}^{c} p_i(t) \cdot \log_2 p_i(t)$
  - $\log_2 0 := 0$
- **Gini Index**: $Gini(t) = 1 - \sum_{i=1}^{c} p_i(t)^2$
- **Classification Error**: $CE(t) = 1 - \max_i[p_i(t)]$

$p_i(t)$: relative frequency of training instances that belong to class i at node t
c: total number of class labels



Min entropy = 0; Max entropy = need to ask many questions for arriving at the right answer to correctly distinguish among classes in case of binary classification.

- *A class of students and a teacher ask questions to the topper student then next time everyone expects he would only answer then entropy is **less**.*
- Most classes are 6 and there are a lot of 1's present then the entropy is **low**.
- "Non-fair dice" (rated) which mostly falls on 6. Entropy => **less**.
- DT is a greedy algorithm that will **not search the solution space**.
- **Info Gain from fair dice to unfair dice is high.**
- For multi-classes, a single question cannot answer (entropy>1).
- **Entropy is only specific for a subset**, it does not take into account the complete picture.
- ID3 prefers a more **general** hypothesis. **NO backtracking.**
- Why do we have this weighted formula in entropy? **To remove the bias.**
- Why do we need IG when we already have entropy? **We need to count the other possible values. Then we shall get the average entropy across all values of that attribute.**
- Why giving maximum weightage to a certain attribute with more value is a problem? **Results in overfitting**
    - **The bigger the weightage the more memory it needs to build the tree.**
    - **A column with all unique values (majority class).**
- **When do you stop the ID3 algorithm?**
    - Reach pure subset
    - Reach the leaf node (homogenous node)
    - Design a threshold -> do not split any further
    - When attribute values are the same.
    - Depth of the tree (level 3-4).

The formal definition of overfitting in general:
Given a hypothesis space *H*, a hypothesis h ∈ H is said to **overfit** the training data if there exists some alternative hypothesis h` ∈ H , such that *h* has a smaller error than *h*' over the training examples, but *h*' has a smaller error than *h* over the entire distribution of instances.
Prevention:

- Don't try to fit all examples, stop before the training set is exhausted.
- Fit all examples then prune the resultant tree.

How does one tell if a given tree is one that has overfit the data?

- Extract a *validation set* not used for training from the training set and use this to check for overfitting. Usually, the validation set consists of one-third of the training set, chosen randomly.
- Then use statistical tests, eg. the chi-squared metric, to determine if changing the tree improves its performance over the validation set.
- A variation of the above is to use MDL to check if modifying the tree increases its MDL with respect to the validation set.

If we use the validation set to guide pruning, how do we tell that the tree is not overfitting the validation set?

In this case, we need to extract yet another set called the *test set* from the training set and use this for the *final check*.

Guarding against bad attribute choices:

The information gain measure has a **bias that favors attributes with many values** over those with only a few.

Obviously, no other attribute can do better. This will result in a very broad **tree of depth 1**.
To guard against this, use ***GainRatio*(*S*, *A*)** instead of *Gain*(*S*, *A*).

"**An overfit DT contains the exact value as the perfect split according to the training examples but, cannot be used for predictions as it satisfies only the training examples and thus will capture less information.**"

- Overfitting in DTs caused by, **more nodes not proportional to class/target, and with more splits, data complexity of underlying distribution, model complexity, perfect split.**
- Building trees that **adapt too much** to the training examples may lead to overfitting.
- Overfitting is more likely with non-parametric and non-linear models (DT, kNN) that have more flexibility when learning a target function.

Additional side notes about DT:

- The whole hypothesis space is reached (target is reached).
- Easy interpretation, complex models like nnets use DTs for explainability.
- Attribute test conditions must be chosen wisely.
- Greedy search (will not search for all the best possible tree alternatives, the one chosen might not always be the best one.)

When is it desirable to consider DTs?

- When instances are describable by **attribute-value pairs <xi, yi>**.
- Target function is discrete-valued.
- Presentation of a **disjunctive hypothesis** may be required.
- Possibly **noisy** training data.

- Variance increases as the complexity increases. So the bias **decreases** as variance **increases. Hence, as the model complexity increases the bias is less and the variance is more.**
- DTs are **non-parametric, unstable (If the learning algorithm is stable, changing input slightly won't put an effect on the hypothesis. In unstable, it would.)** classifiers.
- More no.of nodes hence captures more information that is specific to the training data. **HIGH BIAS == FEWER NODES**.

- Bias in the model proportional to the no.of nodes. More assumptions -> more bias, less assumptions -> less bias -> better fit the training data, hence training error is lower.
- Early stopping leads to **underfitting** not always but mostly.
- **Lower gini impurity, higher is the homogeneity** of nodes. Gini impurity of a pure node is **zero**.

REP (*uses: validation set-unseen unbiased generalized test set*):

- Bottom up approach. Iterative process.
- Pre Pruning is faster than post pruning.
- **You cannot just remove a node without going through all the other nodes**.
- Goal: in every iteration find the best representative tree with the **minimal error** and not optimize it.
- Binary (2-way) split -> high bias; more splits -> low bias
- Smaller tree underfitting. **Beyond 2 iterations it's an indication that the model is not a good model and it will not capture much of the information.**
- Beyond running or performing this algorithm make sure that the entire data is following the **same sample distribution**.
- The difference between info gain & split info is that on the denominator extra info is present. **Split info also considers the size of the splits that you have**.
- Info gain is **biased towards attributes with more values** (multi-valued attribute ex. Age-tends to split for each unique value). Hence, not a good idea.
- Tries to penalize the column with more unique values. The weight of split info is more with the no.of unique values & hence the gain value is less for that. **DENOMINATOR > GAIN RATIO IS LESS**
- Choose the one with the "**perfect split" with the highest ratio. Problems with perfect split: overfitting, computationally complex.**

Summary of the pruning process:

- Each node is a candidate for pruning .
- Nodes are removed only if the resulting tree performs better on the **validation set**.

- Nodes are pruned iteratively: at each iteration the node whose removal most increases accuracy on the validation set is pruned.

Other methods to prevent overfitting in DTs : rule post pruning.

Gain ratio:

- Disregard the class information.
- IG biased towards multi-valued attributes to countermeasure this Gain ratio reduces the **bias**.
- When choosing an attribute it takes into account the **number and size of branches of the tree.**

Problems with Gain ratio:

- May just choose an attribute because its **intrinsic information** is very low. Solution: only consider attributes with **more than average** info gain.

The 'Horizon' problem:

Question: What should be the optimal size of the final tree?

- A tree that is **too large** may seemingly overfit the training data and poorly generalizes to new samples.
- A **small tree** might not capture important structural information about the sample space.
- However, it is hard to tell when a tree algorithm should stop because it is impossible to tell if the addition of a single extra node will dramatically **decrease the error**. This problem is known as the '*horizon effect*'.

Chapter 6

- Perceptron does not overfit because it is not a network of (dense) layers.
- Learning in perceptrons and nnets is regarding **weights including bias.**
- Only learn linear decisions hence, cannot be used for non-linear situations.

- The weights are stopped learning or modifying until we get error = 0.
- Not able to handle XOR function (parity problem) due to presence of nonlinearity.

Can we have a negative learning rate? Yes, if so then update rule must be changed as $\Delta w_i = (t+o)\, x_i$, else we are essentially taking steps away from a point of minima, which is absurd since our goal is to minimize the cost function and thus never learns and the performance decreases at each iteration.

What do we do with the bias in perceptron learning? It can be used as a threshold by converting from equality to inequality.

Neural Networks:

- Primarily used for **classification**, but can also be used for regression by changing the activation to produce a quantity.
- Basic assumption of error: error at output layer is due to the error at hidden layers -> to minimize this we use the bp algorithm.
- Bp is not essentially part of nnets but a way to compute the **gradients**.
- A fully connected nnet is always not necessary (skip connections).

Different terminologies of nodes in nnets:

- At input layer => passive nodes, not actively participating.
- At hidden layer => unobservable outputs, active nodes. They do not know what the ground truth is.
- At output node => observable output.
- ratio - n:n:n, where n = no.of nodes at each layer.
- Classical activation function used: **Sigmoid activation** (S-curve; derivative: gaussian).

Why do we really need a hidden layer? Hidden layer is an integral part of any neural network. A network simply without a hidden layer is as good as classifying linear decision so it may not seem to be a sensible network either. For handling non-linearity and for employing **different**

activation functions and thereby to have an impact on the output node(s) we need a hidden layer. By having non-linearity we make the output a differentiable function of inputs which otherwise was not possible.
Problems with nnets:

- They do not have a memory hence have problems in unlearning and forgetting if some weights have been learnt wrongly it needs to be updated.
- Activation function needs to be chosen as per the problem at hand. Use of a sigmoid can cause '**vanishing gradient**' problems as the gradient becomes very small till it reaches the previous layers. (Solution: to use **skip connections,** skip some connections in the nnets and feed the output of one layer as the input to the next layer. It thereby provides an alternative path for gradient descent with bp).
- Another problem is **exploding gradient**, where the loss turns out as NAN, so to mitigate this we need a **gradient clipping** strategy: -5<=GD<=5.

Why is it not a good practice to initialize all weights to equal values?

If a solution requires that unequal weights need to be developed then nnet can never learn (problem: **Symmetry breaking**).

When should we stop training the nnet or the bp algorithm?

- If you simply stop as soon as your validation or training error gets worse between two subsequent steps, you end up in suboptimal solutions prematurely.
- Network then calculates the errors (variance) on the training data and validation data.
- **Stop training when the validation error is the minimum.** If you stop when training error is minimum then you possibly overfitted and the nnet cannot generalize to unseen data. In most cases the validation error is usually bigger than the training error -> estimate by using cross validation.

Prevent overfitting in nnet: make use of dropout as a regularization technique by dropping a few nodes from a dense network.

Why do you need a zero-centered activation function?

- Zero-centered activations ensure that the mean or average activation value is around 0.
- To represent **neutral states (tanh is a very good example of zero-centered activation function)**.
- This property is important in nnets because it has been empirically shown that models operating on normalized data whether it be inputs or latent (hidden) activations experience faster convergence.

Why do you need sparsity in activations/networks or sparse networks?

- Firstly, sparse networks are **faster** than dense networks.
- Some units are not activated for better generalization (one regularization technique).
- Sparse activations (50% passive nodes; 50% active nodes) make concise models and will have better predictive power and thereby lesser chance of overfitting.

ReLU:

- **Non-linear, non-zero centered, non-differentiable** at zero but differentiable elsewhere.
- Linearity (makes efficient networks) in relu means slope is not **plateau or saturate** which makes searching for gradient values difficult.
- Derivative of relu is either 1 or 0 i.e **unbounded positive infinite.** Hence**,** suffers from **exploding gradients.**
- It fixes the **vanishing gradient problem (from statistics "problem of underflow").**
- It directly deactivates activations for values below 0, therefore activation of units in a network is overall smaller (sparsity property).
- **Death of neurons** happens when the **learning rate is too high or there is a large negative bias**. Basically, no learning for any values below 0, there is no activation whatsoever, neurons might be never activated for any input, if weights are pushed in that direction.

Proof of non-linearity of relu:

- By the property of distributivity, $f(x+y) = f(x) + f(y)$
- Let x = 2 and y = -2
- LHS: relu(-2+2) = 0
- RHS: relu(-2) + relu(-2) = 0 + 2 = 2 therefore LHS≠RHS => hence, not linear.

Comment on usage of relu: Not preferred, due to the inconsistency of results for negative input.

ELU: variant of relu that produces activations instead of letting them to be zero, when calculating the gradient. Still has **exploding gradients** problems.

SELU: faster network convergence, free from **vanishing and exploding gradients**.

Leaky relu:

- Helps solve the vanishing gradient problem, by allowing a small gradient.
- Still non-linear activation, but also derivable below 0.
- **Sparsity of relu is lost in this version**.
- Still suffers from exploding gradient problems.

Randomized Leaky relu:

- Introduces a random negative slope to prevent overfitting.
- Becomes linear when differentiated.

Softmax activation:

- + Able to output more than just binary classification (like sign or sigmoid) i.e. works with multi-class problems.
- + fully derivable, but still has non-linear properties (therefore is a non-linear multinomial classifier).
- + normalized output (between 0 & 1).

- - does not work for multi-label problems (each instance can belong to more than one class).
- - again non-zero centered but, since it is being used mostly as an output layer, it does not matter.

Why is softmax not preferred at the hidden layers but only at the output layer?

Short answer: softmax at the hidden layer decreases the accuracy and speed of learning.

Detailed answer: **"Variable independence"** if softmax is used at the hidden layer then you will keep all your nodes **linearly dependent** which leads to poor generalization (low test accuracy). And thus a lot of regularization effort is required to keep variables independent, uncorrelated and quite sparse.

"**Training issues**" the speed of training becomes very slow. If you want to make the network perform better by employing softmax at the hidden layer then all the activations from the hidden layer are expressed lower. Meaning the average activation on a higher level which might increase the error and harm your training phase.

- It reduces the expressive power of your models.

tanh activation:

- + very similar to sigmoid function in properties.
- + **zero-centered**, therefore neutral states can be shown.
- + Faster takes on high values beyond the zero values therefore it has **stronger gradients**.
- + Has values (range) between -1 and +1. Therefore, mean output is at around zero, whereas, in sigmoid (more like logit) is on average some positive value.
- - computationally more expensive than sigmoid.
- - similar problems as with sigmoid e.g. error gradient is very small near origin i.e., initializations with very **small weights not a good idea**, also suffers from vanishing gradient problems.

Why is sigmoid activations preferred mostly?

- They use exponential functions, and so gradients become linear at some point and help quicker in iteration.
- Sigmoid outputs are **non zero-centered**.

Choosing the right activation and loss function:
Nnets looks for faster training so it is necessary to choose the loss function based on what output you want otherwise, you obtain thrash values and thus it will go into infinite loop.
Regression: one node, linear activation
Binary classification: one node, sigmoid activation
Multi-class: one node per class, softmax activation
Multi-label: one node per class, sigmoid

Backpropagation

Hypothesis space of bp: it is the space of all functions that can be represented by assigning weights to the given, fixed network of interconnected units.

- Bp searches the space of all possible *h* using GD to iteratively reduce the error in network fit to the training examples.
- GD converges to a local minima in the training error wrt the network weights.
- Property of bp is its ability to invent new features that are not explicit in the input to the network.

Neurons are a DAG

Downstream is "**everything after a certain neuron**". You say neuron y, is downstream of neuron x if and only if there is a directed path from x to y. E.g. y is downstream of x if and only if y uses data processed by x.

Neural nets are flexible when it comes to usage (for explainability: the output of nnets can be used with DT.).

Why might the learning process get stuck at a low learning rate?

1. When using sigmoid units: gradients might get very small and when you multiply this small gradient with a low learning rate then the update cannot happen any further as the training cannot escape from that. Thereby the updates done so far **vanish**.
2. General problem: The error function might not have just one local minima but multiple and hence, the training might get stuck.

Problems with high learning rate.

- Oscillating gradients and thus it will not converge due to overstepping (solution: use annealing learning rate).

Learning rate decreases as the iteration increases (programming assignment-1).

To countermeasure low and large lr's we can use **learning rate schedule** or alternatively use **momentum.**

Adam optimizer (first and second momentum for the loss function):

- The best optimizer works on the first and the second derivative of the loss function but the second derivative is relatively very expensive to calculate. So you make some estimations by using the moment.
- Combines properties of RMSprop + momentum.
- Take the moving average of the previous gradient.
- Do not oscillate or get stuck.
- Little memory requirements.
- **Key idea: provision of different lr for different weights.**
- Appropriate for problems with noisy gradients. SGD maintains a single lr for all weight updates and the lr does not change during training.

RNN:

- Used for handling non-sequential data.
- Long term dependencies are not captured
- Windowing might solve having the previous context available specially for long-term dependency.
- Bi-grams and trigrams will NOT help in long-term dependency.
- It works well for smaller sequences but problematic for longer sequences.
- Training RNN happens via unfolding of the network overtime and then by using regular bp.
- This creates a very deep network therefore usually suffers from **vanishing gradient problems** thus making the history unimportant in training.

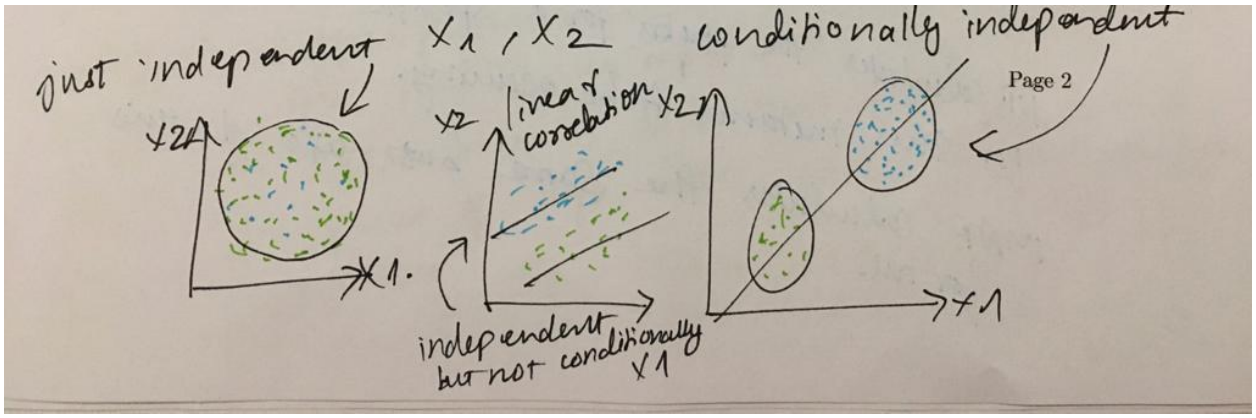
LSTM:

- Lstm tries to solve vanishing gradients by regulating the flow of information from past iteration through '**gating'**.
- Uses the concept of '**cell state**' and hence is able to process longer input sequences.
- Mostly uses tanh and sigmoid activations.

FFNN<RNN<LSTM

Chapter 8

Naive Bayes assumption: The features in the dataset are **conditionally independent given the class**. (short answer: *conditional independence*).



- Independent and identically distributed (i.i.d).
- NB checks for **each individual instance** the most probable class (most similar hypothesis). **P(D|h).P(h)** => collectively called as '**potential**'.
- MAP calculates for the whole dataset. i.e. how likely is this hypothesis true if we look at the **whole dataset** and then we take the maximum out of this. **Hypothesis that maximizes the product of prior and likelihood.**
- MLE is a special case of MAP where our prior is uniform (all values are equally likely).
    - Limitation of MLE:
        - It assumes that the dataset is **complete or fully observable.** This does not mean that the model has access to all data; instead it assumes that all variables relevant to the problem are present.
- **Handling missing values**
    - Take the maximum of the class labels and assign the majority in case of categorical (class imbalance).
    - Take the mean of the values in case of numeric (affected by outliers).
    - Median values in case of numeric (better option).
- Bayes optimal classifiers can be expensive if there are many hypotheses (solution: use **Gibbs classifier**).

Why is size a parameter for probability estimation is spam classification?

Spam emails are lengthy. It might be interesting to see the average size of the spam. Only words which come very frequently or seldomly do not contribute too much to the prediction/classification.

Applications of NB in practical life:

Loan: The maximum support of 50,000 euros per film per country can be granted as a conditionally repayable loan, whose repayment depends on the success of the film in the cinema. Depending on this, classify as a defaulted borrower or not.

Chapter 9

Assumption in KNN:

- A **majority vote among the nearest neighbors** to generate a class label.
- Examples with **similar features will yield similar kinds of predictors**.
- Works in 2 modes:
  - **Online mode**: classification time
  - **Offline mode**: in memory (training phase) - if all the examples are stored in memory it will run **slowly**.
- Target may be either **discrete-valued or real-valued.**
- **Inductive bias: "classification of an instance query will be most similar to the classification of other instances that are nearby in euclidean distance."**
- Lazy learners - delays the processing until a new instance must be classified.
  - +Instead of estimating the target function once for the entire instance space, these methods can estimate it locally and differently for each new instance to be classified.
- Misleading majority class label problems
  - **target/class imbalance problem**: model gets more exposed to learn from majority since it dominates in comparison to minority class. Model is more biased towards the majority class.
  - **Border-line cases (near the decision boundary)**: cases lie on the border of the decision boundary and the closest points and the others are far away. (Solution: Summation of weights will be calculated).
- If k in knn is **even** then there would be a tie in classifying the instances.
- **Should not use knn when time is a KPI**.
- Easier to work with **categorical variables as a classifier**.
- + robust to noisy data.
- + effective when it is provided with a **sufficiently large set of training data**.

- kNN **does not explicitly** compute decision boundaries.
- kNN is **non-parametric** - there is no assumption for underlying data distribution. Meaning model structure is determined from the dataset.

To solve the majority class label problem

- In a majority classifier all those summations of weights would be 1. But if not 1 then they just take the inverse distances, then points with closer have higher impact on the end results than farther away points.
- Alternative: for each class that you have the neighbors you sum up the weights. Whoever has the highest “sum of weights” this will be the class that you will assign to (winner).
- Alternative: class distribution + distance metric.

Weighting scheme:

- **Distance weighting** - classification to the function with the maximum value is assigned.
- **Attribute weighting** - to solve problem of ‘**curse of dimensionality**’
- **Class-based weighting** - takes into account the instance of each class. It penalizes the miss rate by minority class by setting a higher class weight and at the same time reducing weight for the majority class. During training, we give more weightage to the minority class in the cost function so that it would provide a higher penalty to the majority class and the algorithm could focus on reducing the errors for the minority class.

LWR form of kNN:

- Local - function approximation based only on data near the query point.
- Weighted - weighted contribution of each training example is weighted by its distance from a query point.
- Uses a nearby or distance-weighting training sample.
- LWR uses the **same distance functions** as knn classifier.

Distance functions in knn: **L1, L2, minkowski** for **continuous** attributes. **Hamming distance** for **categorical** attributes.

Global method: using all training samples when classifying a new query.
Local method: only the nearest training samples are considered (classification time).
Instance-based approximates differently (locally) to the target function for each distinct query instance that must be classified.
Cons:

- The cost of classifying new instances can be high because all computations (determining neighbors) take place at **classification time** rather than when the training examples are first encountered.
- Consider **all attributes of the instances (affected when highly irrelevant attributes are present)** when attempting to retrieve similar training examples from memory (**slower training phase**).

Drawback of kNN:

- It suffers from '**curse of dimensionality**' when highly irrelevant attributes are present (solution: use attribute-weighting method).
- **Efficient memory indexing** (solution: use kd-tree. Instances are stored at leaves of trees, nearby instances stored at the same or nearby nodes. Internal nodes of the tree sort the new query to the relevant leaf by testing selected attributes of the query point.)

Failure cases of kNN:

- When the query point is far away from the peer data points.
- If we have jumble/mixed datasets.
- High memory requirement as it has to store all the data points.
- Prediction stage is very costly.
- Sensitive to outliers, accuracy is impacted by noise or irrelevant data.

Optimal k in knn is estimated by means of **cross-validation** or AIC, BIC.

kNN	Decision trees
Selects "all" attributes	Selects "subset" of attributes.
Lazy learner	*Unstable. Eager learner*

RBF:

- kernel function is the function of distance that is used to determine the weight of each training sample. Choose a set of kernel functions **smaller** than the number of training examples.
- Choice of kernel functions allows RBF networks to fit the training data exactly.
- RBF networks are typically trained in a two-stage process.
- Reason: kernel functions are fixed during stage 1 and in stage 2 the linear weight values can be trained efficiently.

Limitations of kd-tree:

- It runs in **O(log n)** average time per search in a reasonable model.
- Storage for the kd-tree is **O(n)**.
- Assuming 'd' is small the preprocessing time is **O(n log n)**.

Conclusion: kd-tree works well on data which is **uniformly distributed** and the **dimension is small**.

Local approximators: kNN, LWR, CBR {lazy learners}.
Global approximators: RBF, BP, C4.5(DT) {eager learners}

Issues with CBR (ex: issuing VISA to a resident from the aliens office.):

- To develop methods for indexing cases.
- Syntactic similarity measures (in word sequences).
- Examples in recommendation systems.

Cover tree: a data structure used for partitioning of the metric spaces to speed up operations like nearest neighbor or range searches.

Active learning:

- Oracle can be anyone not necessarily humans, machines can also be possible.

- Case base in AL: Contains/holds the most informative instance to be labelled by the experts (border-line cases or anything else).
- Instances near to the decision boundary may not be the most informative instances because they do not represent the distribution purely.
- Can be very expensive also but stores the complete set of data (training + testing).

IB1:

- Has all the cases in the case base.
- Identical to the kNN algorithm.
- Also performs normalization of attribute ranges similar to kNN.
- Processes instances incrementally, unlike traditional NN algorithms.
- Practical issue: the case base grows quickly.

IB2:

- Stores only misclassified instances from the training set.
- Order dependent on how instances arrive.
- Low noise tolerance, low memory demands.
- IB2's classification accuracy **decreases** more quickly than IB1's as the noise level increases.
- Reason: noisy attributes are more likely to be misclassified and IB2 saves only those, which it uses to generate classification decisions.
- Assumption: vast majority of misclassified instances are near-boundary instances that are located in a small neighborhood of the boundary (border-line cases).
- Classification performance of IB2 is only marginally worse than IB1.

Q) IB2 stores which misclassified instances? Training, testing or both?
Ans: stores just the training examples in the CB.

IB3:

- "**Wait and see**"
- Includes only misclassified cases in current CB and removes "**bad cases**".

- "**Evidence gathering method**" to determine which saved instances are expected to perform well during classification.
- Maintains a **classification record** with each saved instance.
- A classification record saves an instance's classification performance on subsequently presented training instances and suggests how it will perform in future.
- Uses **significance testing** to determine which instances are good classifiers and which ones are believed to be noisy. The former are used to classify subsequently presented instances. The latter are discarded from the concept description.

Supervised learning task: to find a **mapping function** from input to output.
Unsupervised learning task: without the labels **map** instances to groups.
Supervised learning experience: target + instances.
Unsupervised learning experience: instances.

Chapter 10 (clustering)

k-means :

- Stopping criteria: "**until mapping of the instances to the cluster centers do not change**".
- Flat clustering (**Hard assignment** of points to groups).
- Requirements: the value 'k' for the number of clusters and a distance metric (euclidean).
- **Non-deterministic** approach: each time with different iterations and centroids you get different clusters.
- It prefers "**cloud-like**" clusters. Suffers from local optima (elongated, strangled clusters).
- A quick and easy way to find **groups** in the data. Assumes the cluster are **independent**.
- Favours clusters of **same density, same size, shape.**
- **Affected by outliers.**

Applications : cancer subtype prediction from "gene expression patterns" based on tissue samples from healthy individuals to sick.

How to determine the optimal number of k-clusters?

Direct methods: elbow method (**knee locator** in python), average silhouette method (a disadvantage of elbow and average silhouette method is that they measure a "**global clustering characteristic**" only).
Statistical method: Gap statistic (better estimate).
Heuristic methods: **model scoring and selection**: AIC (puts emphasis on model performance, lower values are preferred) or BIC (penalizes for size of your data), cross validation by using grid search.

HAC (deterministic):

- No need for k but need to have a **distance measure** for points and clusters. Look for **non-spherical** shaped clusters.
- Stopping criteria: "**stops when only one point is left for merging**".
- Visualizations by using **dendrogram** (perform a cut to obtain clusters at each level. Note: cut need not be horizontal).
- Each point is initialized as one cluster.
    - MIN / SLINK
        - Produces **chain-like** structures: a pair forms, then an object rejoins the pair and so on. The resulting dendrogram does not clearly show the separated groups, but can be used to identify **gradients** in the data.
        - Similarity is most similar to members and is **monotonically decreasing** from iteration to iteration.
        - It is **optimal** clustering.
        - Merging criteria: **local**
        - used to identify outliers or affected outliers.
        - It can be used with similarity or dissimilarity measures.
        - It is **continuous** from weak clustering to the strong clustering. Hence, this method is invariant to monotone transformations of input data.
    - MAX / CLINK
        - Derives **compact** clusters.
        - Tends to produce small groups separately.
        - Good for looking for discontinuities in data.

        - Merging criteria: **non-local,** thus the entire structure of the clustering can influence merge decisions.
        - Since it is **sensitive to outliers** it causes **less-than** optimal merging hence, clustering is not optimal.
    - Centroid linkage
        - It is **non-monotonic**. **Similarity can increase** during clustering (**inversion property**).
        - Not affected by noise, but have a bias towards finding "**global patterns**".
- EM algorithm:
    - Set of instances + missing attribute and looks for the most probable model for the data.
    - Determine based on the model the missing attribute.
- Complexity of HAC algorithms:

MIN < MAX < AVG < CEN < WARDS

k-means	HAC
Handles big data.	Requires to work with concise data.
Time complexity: linear O(n)	*Time complexity: quadratic* $O(n^2)$

FCM:

- Soft clustering technique.
- Stopping criteria: "**stops when membership values remain the same or until the SSE reduced to a certain threshold**".
- A point can belong to **more than one cluster**.
- Uses membership values. And its **summation of it cannot be >1 and each individual membership value cannot be less than 0 (non-negative).**
- Uses a hyperparameter 'm' i.e. the **fuzzifier - it controls the amount of overlap.**
- K-means is a very special case of FCM and can be converted back to k-means by putting 1(hard assignment) where it approximates to 1 and 0 (hard assignment) where it approximates to 0.

LVQ:

- Fixed lr might cause **oscillations**.
- **Parameterized clustering**.
- Unsupervised + semi-supervised.
- LVQ requires a partial dataset, whereas k-means full dataset.
- Uses **codebook vectors**.
- Weak clustering. More problems due to iterative nature.
- Optimization results for LVQ and k-means are the same.

SHAP values - (an acronym from **SHapley Additive exPlanations** **https://www.kaggle.com/dansbecker/shap-values**) gives a breakdown of a prediction to show the impact of each feature.

Where could you use this?

1. A model says a bank shouldn't loan someone money, and the bank is legally required to explain the basis for each loan rejection.
2. A healthcare provider wants to identify what factors are driving each patient's risk of some disease so they can directly address those risk factors with targeted health interventions.

Odds ratio: https://senguptasresearchacademy.com/odds-ratio/
Odds ratio is a very effective way of determining association between two variables, mostly influence of one factor on the outcome of interest. If strong enough, and the statistical analysis robust enough, it can even determine causality i.e. prove a cause – effect relationship between a risk factor and disease or an adverse effect and a drug.

FSK rating (Germany) - 0, 6, 12, 16, 18 (multiclass problem).