
ADAX

Python ML Training Others

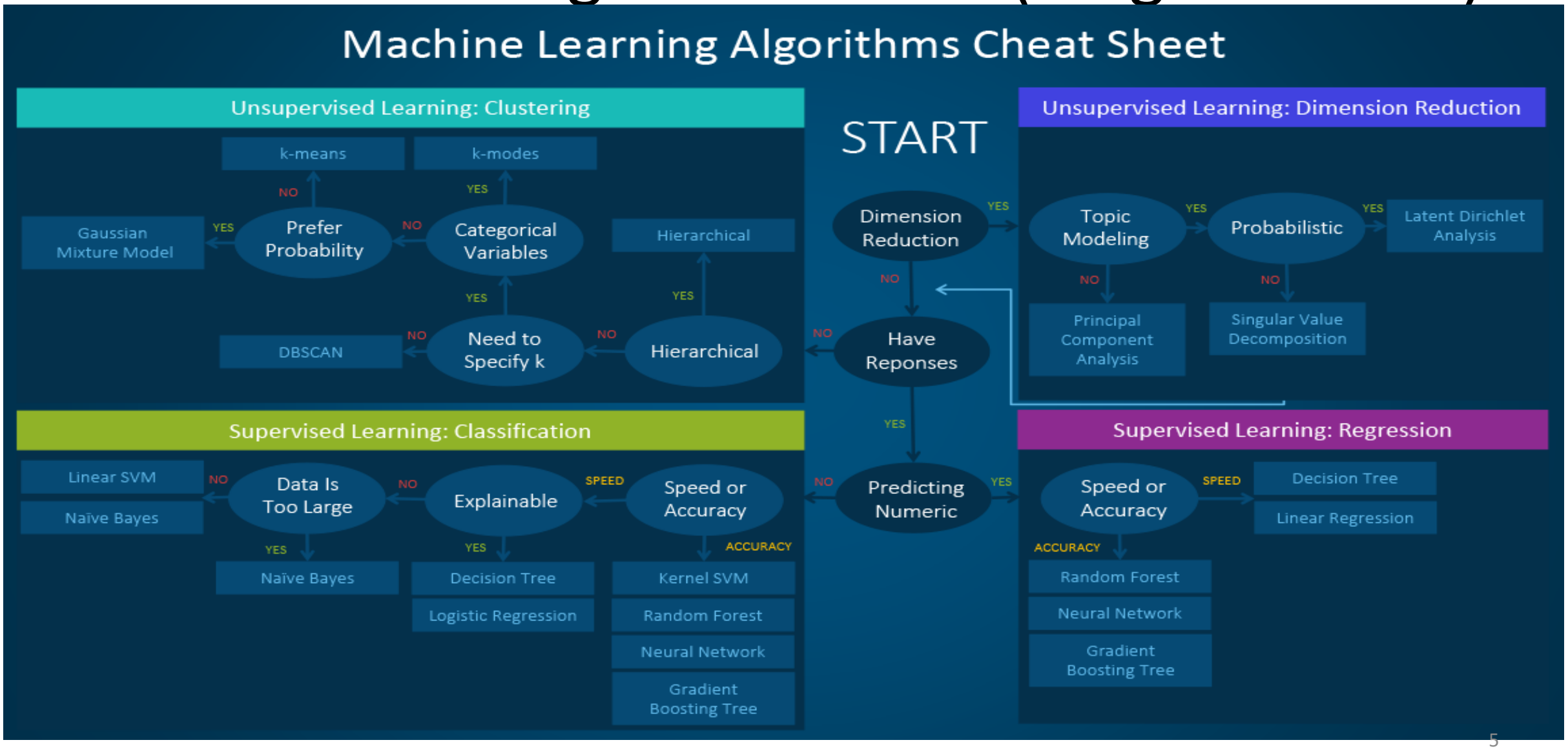
Introduction

- Peter Ho, PhD
- Email : peyter@gmail.com
- Training experience:
  - MDeC Coursera trainer for all cohorts
  - Trainer for Intel ,JPA, MAMPU, TM, Unisel, Unitar
- Consulted on data science projects with TM
- Credit goes to machinelearningmastery.com for succinct explanations

Training Schedule: Day 1

Time	Details
0900 – 0940	k-Nearest Neighbor
0940 – 1040	Logistic Regression
1040 – 1100	**Morning Break**
1100 – 1200	Naïve Bayes
1200 – 1300	Support Vector Machine
1300 – 1400	**Lunch**
1400 – 1540	Decision Trees
1540 – 1600	**Afternoon Break**
1600 – 1700	Hands on exercises

Machine Learning Cheat Sheet (blogs.sas.com)



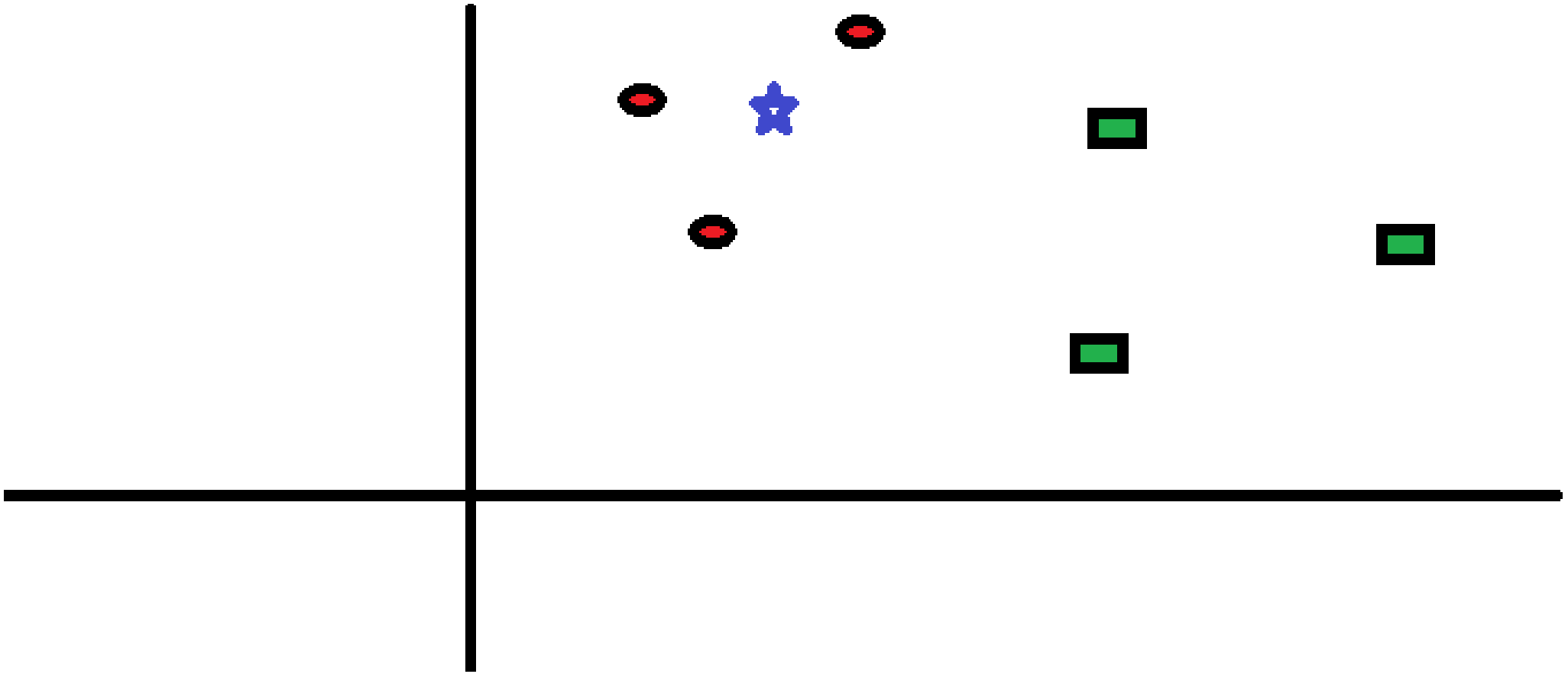
k-Nearest Neighbor (kNN)

- 0900 – 0940

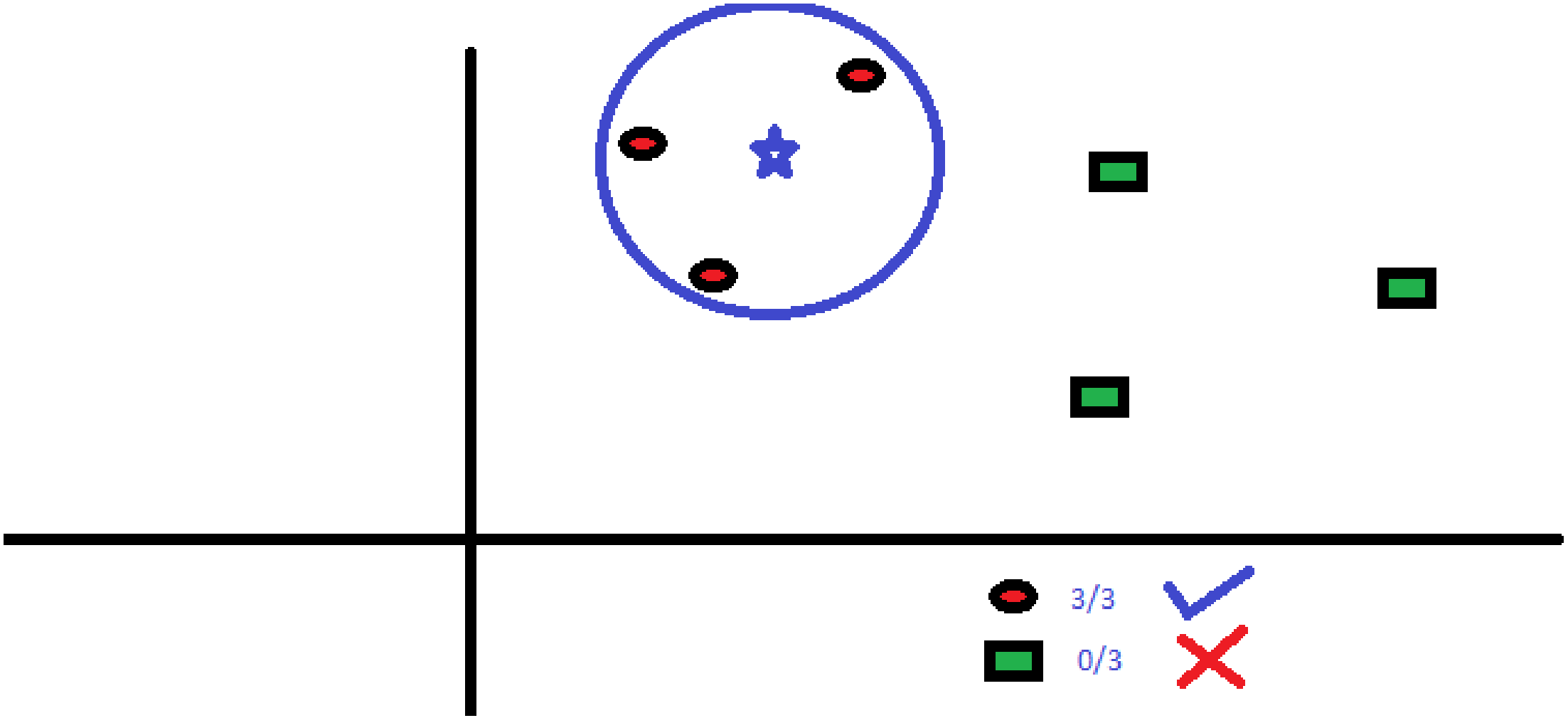
k-Nearest Neighbour

Advantages	Easy to interpret output, low calculation time
Disadvantage	1. Computational complexity increases when number of observation increases
Classification tips	1. For even number of classes, choose an odd number of K. Vice versa, for odd number of classes, choose an even number of K . This prevents a tie from occurring 2. The optimum number of K can be calculated through the use of grid-search techniques 3. In lieu of grid search, we can start by iterating from 1-NN to 21-NN to get best results
Data preprocessing steps	1. Rescale data to [0,1] 2. Standardize the data distribution if it is Gaussian 3. Address missing data. Missing data means distance between samples are not able to be calculated and that observation will be dismissed 4. Reduce dimensionality
Regression	1. k-NN can be used to perform regression as the prediction is based on the mean or the median of the K-most similar instances.

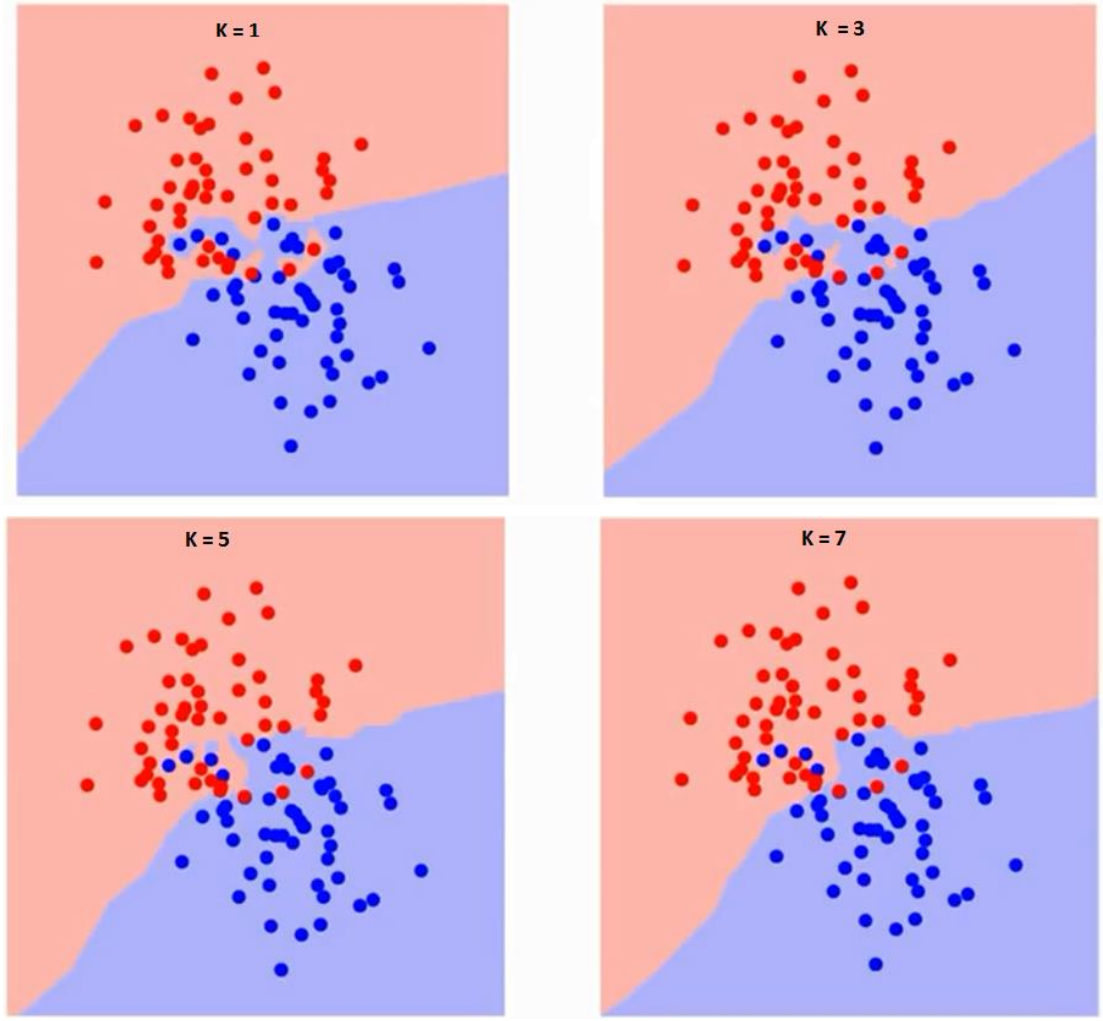
k-Nearest Neighbour



k-Nearest Neighbour



Effects of choosing different values for K



Pseudo code for KNN

1. Load the data
2. Initialise the value of k
3. For getting the predicted class, iterate from 1 to total number of training data points
    I. Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
    II. Sort the calculated distances in ascending order based on distance values
    III. Get top k rows from the sorted array
    IV. Get the most frequent class of these rows
    V. Return the predicted class

Logistic Regression

- 0940 – 1040

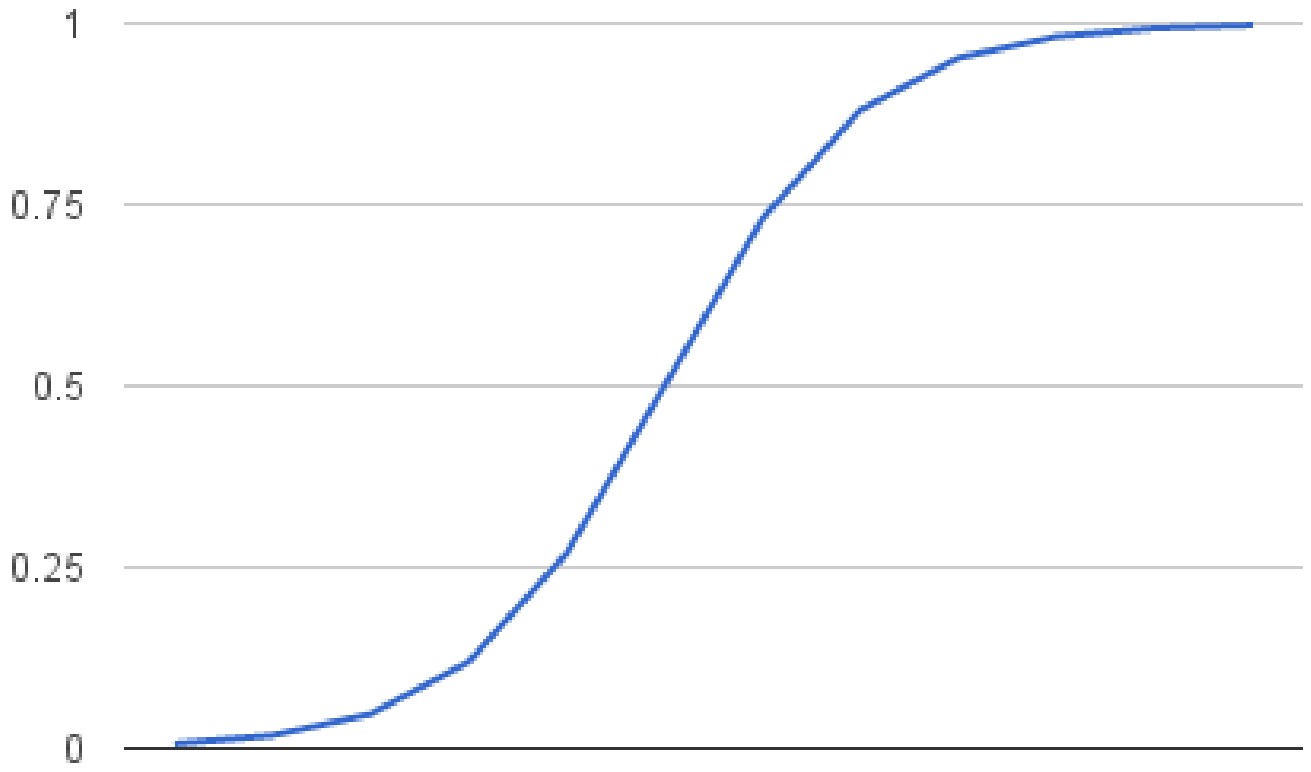
Logistic regression

Advantages	**1. Good for problems with low signal to noise ratio** **2. Strong for binary class problems**
Disadvantage	1. Possibility that the MLE algorithm fails to converge dues to overly sparse data, or when the data is too highly correlated

Logistic Regression

- Named after the logistic function
- The logistic function (aka sigmoid function) is an S-shaped curved that can map any real valued number into a value of between 0 and 1 (but never at the limits)
- The formulae is 1/(1+e^-value), where e is the base of the natural logarithm, and the value is the value that we wish to transform

Logistic Function



Representation of the Logistic Regression

- Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y).
- A key difference from linear regression is that the output value being modeled is a **binary values (0 or 1)** rather than a numeric value.
- Below is an example logistic regression equation:

$$y = e^{(b0 + b1*x)} / (1 + e^{(b0 + b1*x)})$$

The assumption here is made that y has no noise

- The actual representation of the model that you would store in memory or in a file are the coefficients in the equation (the beta value or b's).

Logistic Regression and Probability

- Logistic regression models the probability of the default class (aka the first class)
- If we are modeling people's sex as male or female from their height, then the first class could be male and the logistic regression model could be written as the probability of male given a person's height, or more formally:

$$P(\text{sex=male}|\text{height})$$

- Written another way, we are modeling the probability that an input (X) belongs to the default class (Y=1), we can write this formally as:

$$P(X) = P(Y=1|X)$$

Logistic Regression and Probability

- By substituting the previous example equation, we may state our LR model as

$$p(X) = e^{(b0 + b1*X)} / (1 + e^{(b0 + b1*X)})$$

- The above equation can be converted into the following by removing the term e

$$\ln(p(X) / 1 – p(X)) = b0 + b1 * X$$

- Now the right hand side of the equation is linear in nature while the left hand side is the **probability of the default class**
- The ratio on the left hand side of the equation is the **odds of the default class, therefore the lefthand side is the log-odds or probit**

$$\ln(odds) = b0 + b1 * X$$

- Finally moving the exponent back to the right hand side yields

$$odds = e^{(b0 + b1 * X)}$$

- Therefore the LR model is indeed a linear combination of the inputs, in relation to the log odds of the default class

Learning the LR model

- The beta values b0 and b1 are learned from the training data, where algorithms such as maximum-likelihood estimation is used to predict a value of 1 for the default class and a value of zero for the other class.

Predictions using the LR model

- Let's say we have a model that can predict whether a person is male or female based on their height (completely fictitious). Given a height of 150cm is the person male or female.
- We have learned the coefficients of b0 = -100 and b1 = 0.6. Using the equation above we can calculate the probability of male given a height of 150cm or more formally P(male|height=150). We will use EXP() for e, because that is what you can use if you type this example into your spreadsheet:

$$y = e^{(b0 + b1*X)} / (1 + e^{(b0 + b1*X)})$$

$$y = \exp(-100 + 0.6*150) / (1 + EXP(-100 + 0.6*X))$$

$$y = 0.000045397868 7$$

- Or a probability of near zero that the person is a male.
- In practice we can use the probabilities directly. Because this is classification and we want a crisp answer, we can snap the probabilities to a binary class value, for example:

$$0 \text{ if } p(male) < 0.5$$

$$1 \text{ if } p(male) >= 0.5$$

Data Preprocessing for LR

- Data preprocessing steps for using an LR model include
  - Ensuring Binary Output variable
  - Removal of outliers, as the LR model assumes y has no noise
  - Mapping data to a Gaussian distribution
  - Remove correlated input

Naïve Bayes

- 1100 - 1200

Naïve Bayes

Advantages	**1. Fast , low computational time** **2. Less training data** **3. Scales well** **4. Support binary and multi class problems** **5. Can support probabilistic predictions instead of a hard class prediction** **6. Support continuous and discrete data** **7. Less sensitive to missing data and irrelevant features**
Disadvantage	1. Assumes that features are independent of each other 2. Does not do well when data is scarce 3. Binning continuous features may result in information loss

Bayes Theorem

- In machine learning we are often interested in selecting the best hypothesis (h) given data (d).
- In a classification problem, our hypothesis (h) may be the class to assign for a new data instance (d).
- One of the easiest ways of selecting the most probable hypothesis given the data that we have that we can use as our prior knowledge about the problem. Bayes' Theorem provides a way that we can calculate the probability of a hypothesis given our prior knowledge.
- Bayes' Theorem is stated as:

$$P(h|d) = (P(d|h) * P(h)) / P(d)$$

- Where
- **P(h|d)** is the probability of hypothesis h given the data d. This is called the posterior probability.
- **P(d|h)** is the probability of data d given that the hypothesis h was true.
- **P(h)** is the probability of hypothesis h being true (regardless of the data). This is called the prior probability of h.
- **P(d)** is the probability of the data (regardless of the hypothesis).

Maximum a posteriori

- After calculating the posterior probability for a number of different hypotheses, you can select the hypothesis with the highest probability. This is the maximum probable hypothesis and may formally be called the maximum a posteriori (MAP) hypothesis.
- This can be written as:

$$MAP(h) = max(P(h|d))$$

or

$$MAP(h) = max((P(d|h) * P(h)) / P(d))$$

or

$$MAP(h) = max(P(d|h) * P(h))$$

Normalizing term

- The P(d) is a normalizing term which allows us to calculate the probability. We can drop it when we are interested in the most probable hypothesis as it is constant and only used to normalize.
- Back to classification, if we have an even number of instances in each class in our training data, then the probability of each class (e.g. P(h)) will be equal. Again, this would be a constant term in our equation and we could drop it so that we end up with:

$$MAP(h) = \max(P(d|h))$$

Naïve Bayes classifier

- Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values.
- It is called *naive Bayes* or *idiot Bayes* because the calculation of the probabilities for each hypothesis are simplified to make their calculation tractable. Rather than attempting to calculate the values of each attribute value P(d1, d2, d3|h), they are assumed to be conditionally independent given the target value and calculated as P(d1|h) * P(d2|H) and so on.
- This is a very strong assumption that is most unlikely in real data, i.e. that the attributes do not interact. Nevertheless, the approach performs surprisingly well on data where this assumption does not hold.

Representing a Naïve Bayes Model

- A list of probabilities are stored to file for a learned naive Bayes model. This includes:
- **Class Probabilities**: The probabilities of each class in the training dataset.
- **Conditional Probabilities**: The conditional probabilities of each input value given each class value.

Learning a Naïve Bayes model from the data

- Learning a naive Bayes model from your training data is fast.
- Training is fast because only the probability of each class and the probability of each class given different input (x) values need to be calculated. No coefficients need to be fitted by optimization procedures.

Calculating class probablities

- The class probabilities are simply the frequency of instances that belong to each class divided by the total number of instances.
- For example in a binary classification the probability of an instance belonging to class 1 would be calculated as:
- P(class=1) = count(class=1) / (count(class=0) + count(class=1))
- In the simplest case each class would have the probability of 0.5 or 50% for a binary classification problem with the same number of instances in each class.

Calculating Conditional Probablity

- The conditional probabilities are the frequency of each attribute value for a given class value divided by the frequency of instances with that class value.
- For example, if a "*weather*" attribute had the values "*sunny*" and "*rainy*" and the class attribute had the class values "*go-out*" and "*stay-home*", then the conditional probabilities of each weather value for each class value could be calculated as:
- P(weather=sunny|class=go-out) = count(instances with weather=sunny and class=go-out) / count(instances with class=go-out)
- P(weather=sunny|class=stay-home) = count(instances with weather=sunny and class=stay-home) / count(instances with class=stay-home)
- P(weather=rainy|class=go-out) = count(instances with weather=rainy and class=go-out) / count(instances with class=go-out)
- P(weather=rainy|class=stay-home) = count(instances with weather=rainy and class=stay-home) / count(instances with class=stay-home)

Making predictions using a NB model

- Given a naive Bayes model, you can make predictions for new data using Bayes theorem.

$$MAP(h) = \max(P(d|h) * P(h))$$

- Using our previous example, if we had a new instance with the *weather* of *sunny*, we can calculate:

$$\text{go-out} = P(\text{weather=sunny}|\text{class=go-out}) * P(\text{class=go-out})$$
$$\text{stay-home} = P(\text{weather=sunny}|\text{class=stay-home}) * P(\text{class=stay-home})$$

- We can choose the class that has the largest calculated value. We can turn these values into probabilities by normalizing them as follows:

$$P(\text{go-out}|\text{weather=sunny}) = \text{go-out} / (\text{go-out} + \text{stay-home})$$
$$P(\text{stay-home}|\text{weather=sunny}) = \text{stay-home} / (\text{go-out} + \text{stay-home})$$

- If we had more input variables we could extend the above example. For example, pretend we have a "*car*" attribute with the values "*working*" and "*broken*". We can multiply this probability into the equation.
- For example below is the calculation for the "go-out" class label with the addition of the car input variable set to "working":

$$\text{go-out} = P(\text{weather=sunny}|\text{class=go-out}) * P(\text{car=working}|\text{class=go-out}) * P(\text{class=go-out})$$

Representation for Gaussian NB

- Above, we calculated the probabilities for input values for each class using a frequency. With real-valued inputs, we can calculate the mean and standard deviation of input values (x) for each class to summarize the distribution.
- This means that in addition to the probabilities for each class, we must also store the mean and standard deviations for each input variable for each class.

Learning Gaussian NB Model from Data

- This is as simple as calculating the mean and standard deviation values of each input variable (x) for each class value.

$$mean(x) = 1/n * sum(x)$$

- Where n is the number of instances and x are the values for an input variable in your training data.
- We can calculate the standard deviation using the following equation:

$$standard\ deviation(x) = sqrt(1/n * sum(xi-mean(x)^2))$$

- This is the square root of the average squared difference of each value of x from the mean value of x, where n is the number of instances, sqrt() is the square root function, sum() is the sum function, xi is a specific value of the x variable for the i'th instance and mean(x) is described above, and ^2 is the square.

Making predictions using a Gaussian NB Model

- Probabilities of new x values are calculated using the Gaussian Probability Density Function(PDF).
- When making predictions these parameters can be plugged into the Gaussian PDF with a new input for the variable, and in return the Gaussian PDF will provide an estimate of the probability of that new input value for that class.

  pdf(x, mean, sd) = (1 / (sqrt(2 * PI) * sd)) * exp(-((x-mean^2)/(2*sd^2)))
- Where pdf(x) is the Gaussian PDF, sqrt() is the square root, mean and sd are the mean and standard deviation calculated above, PI is the numerical constant, exp() is the numerical constant e or Euler's number raised to power and x is the input value for the input variable.
- We can then plug in the probabilities into the equation above to make predictions with real-valued inputs.
- For example, adapting one of the above calculations with numerical values for weather and car:

  go-out = P(pdf(weather)|class=go-out) * P(pdf(car)|class=go-out) * P(class=go-out)

Data preprocessing for NB

- Categorical inputs – inputs are expected to be binary, categorical or nominal
- Gaussian inputs – if real valued inputs are given, a Gaussian distribution is assumed. Performed a transform so that input data is Gaussian
- Problems – NB can solve binary and multiclass classification
- Log probabilities – Perform log transform of probabilities to prevent underflow of numerical precision
- Kernel functions – Try using more complex distribution apart from Gaussian
- Update – update the probability of the model when new data becomes available

Support Vector Machine

- 1200 – 1300

SVM

Advantages	**1. A good option when understanding of the underlying data is limited** **2. Works well with both structured and semi structured data** **3. Does not solve for local optima** **4. Scales well to high dimensional data** **5. Less risk of overfitting**
Disadvantage	1. Difficult to choose good kernel functions 2. Long training time for large datasets 3. Difficult to understand and explain the model, variable weights and individual impacts of weights 4. Hard to perform small calibration
Classification tips	1. Optimum values for hyper parameters of SVM can be found using grid search

Maximal-Margin Classifier

- The Maximal-Margin Classifier is a hypothetical classifier that best explains how SVM works in practice.
- The numeric input variables (x) in your data (the columns) form an n-dimensional space. For example, if you had two input variables, this would form a two-dimensional space.
- A hyperplane is a line that splits the input variable space. In SVM, a hyperplane is selected to best separate the points in the input variable space by their class, either class 0 or class 1. In two-dimensions you can visualize this as a line and let's assume that all of our input points can be completely separated by this line. For example:

$$B0 + (B1 * X1) + (B2 * X2) = 0$$

- Where the coefficients (B1 and B2) that determine the slope of the line and the intercept (B0) are found by the learning algorithm, and X1 and X2 are the two input variables.

Calculations

- You can make classifications using this line. By plugging in input values into the line equation, you can calculate whether a new point is above or below the line.
- Above the line, the equation returns a value greater than 0 and the point belongs to the first class (class 0).
- Below the line, the equation returns a value less than 0 and the point belongs to the second class (class 1).
- A value close to the line returns a value close to zero and the point may be difficult to classify.
- If the magnitude of the value is large, the model may have more confidence in the prediction.
- The distance between the line and the closest data points is referred to as the margin. The best or optimal line that can separate the two classes is the line that as the largest margin. This is called the Maximal-Margin hyperplane.
- The margin is calculated as the perpendicular distance from the line to only the closest points. Only these points are relevant in defining the line and in the construction of the classifier. These points are called the support vectors. They support or define the hyperplane.
- The hyperplane is learned from training data using an optimization procedure that maximizes the margin.

Soft Margin Classifier

- In practice, real data is messy and cannot be separated perfectly with a hyperplane.
- The constraint of maximizing the margin of the line that separates the classes must be relaxed. This is often called the soft margin classifier. This change allows some points in the training data to violate the separating line.
- An additional set of coefficients are introduced that give the margin wiggle room in each dimension. These coefficients are sometimes called slack variables. This increases the complexity of the model as there are more parameters for the model to fit to the data to provide this complexity.
- A tuning parameter is introduced called simply C that defines the magnitude of the wiggle allowed across all dimensions. The C parameters defines the amount of violation of the margin allowed. A C=0 is no violation and we are back to the inflexible Maximal-Margin Classifier described above. The larger the value of C the more violations of the hyperplane are permitted.
- During the learning of the hyperplane from data, all training instances that lie within the distance of the margin will affect the placement of the hyperplane and are referred to as support vectors. And as C affects the number of instances that are allowed to fall within the margin, C influences the number of support vectors used by the model.
- The smaller the value of C, the more sensitive the algorithm is to the training data (higher variance and lower bias).
- The larger the value of C, the less sensitive the algorithm is to the training data (lower variance and higher bias).

Kernels

- The SVM algorithm is implemented in practice using a kernel.
- The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra, which is out of the scope of this introduction to SVM.
- A powerful insight is that the linear SVM can be rephrased using the inner product of any two given observations, rather than the observations themselves. The inner product between two vectors is the sum of the multiplication of each pair of input values.
- For example, the inner product of the vectors [2, 3] and [5, 6] is 2*5 + 3*6 or 28.
- The equation for making a prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

$$f(x) = B0 + sum(ai * (x,xi))$$

- This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients B0 and ai (for each input) must be estimated from the training data by the learning algorithm.

Linear Kernel

- The dot-product is called the kernel and can be re-written as:

$$K(x, xi) = sum(x * xi)$$

- The kernel defines the similarity or a distance measure between new data and the support vectors. The dot product is the similarity measure used for linear SVM or a linear kernel because the distance is a linear combination of the inputs.
- Other kernels can be used that transform the input space into higher dimensions such as a Polynomial Kernel and a Radial Kernel. This is called the Kernel Trick.
- It is desirable to use more complex kernels as it allows lines to separate the classes that are curved or even more complex. This in turn can lead to more accurate classifiers.

Polynomial

- Instead of the dot-product, we can use a polynomial kernel, for example:

$$K(x,xi) = 1 + sum(x * xi)^d$$

- Where the degree of the polynomial must be specified by hand to the learning algorithm. When d=1 this is the same as the linear kernel. The polynomial kernel allows for curved lines in the input space.

Radial Kernel

- Finally, we can also have a more complex radial kernel. For example:

$$K(x,xi) = \exp(-gamma * sum((x - xi^2))$$

- Where gamma is a parameter that must be specified to the learning algorithm. A good default value for gamma is 0.1, where gamma is often 0 < gamma < 1. The radial kernel is very local and can create complex regions within the feature space, like closed polygons in two-dimensional space.

Learning a SVM model

- The SVM model needs to be solved using an optimization procedure.
- You can use a numerical optimization procedure to search for the coefficients of the hyperplane. This is inefficient and is not the approach used in widely used SVM implementations like LIBSVM. If implementing the algorithm as an exercise, you could use stochastic gradient descent.
- There are specialized optimization procedures that re-formulate the optimization problem to be a Quadratic Programming problem. The most popular method for fitting SVM is the Sequential Minimal Optimization (SMO) method that is very efficient. It breaks the problem down into sub-problems that can be solved analytically (by calculating) rather than numerically (by searching or optimizing).

Data preparation for SVM

- Numerical inputs – SVM assumes inputs are numerical. If there's categorical inputs convert them into binary dummy variables
- Binary classification – Basic SVM does well for binary problems, although there are extensions for regression and multiclass problem
- Normalization of data – SVM works well when data are normalized/scaled to between [0,1] or [1,-1]

Decision tree

- 1200 – 1300

Decision tree

Advantages	**1. Little to no data preparation necessary** **2. Implicitly perform feature selection** **3. Non linear relationship between features do not affect tree performance** **4. Easy to interpret and explain**
Disadvantage	1. Difficult to choose good kernel functions 2. Long training time for large datasets 3. Difficult to understand and explain the model, variable weights and individual impacts of weights 4. Hard to perform small calibration
Classification tips	1. Optimum values for hyper parameters of SVM can be found using grid search

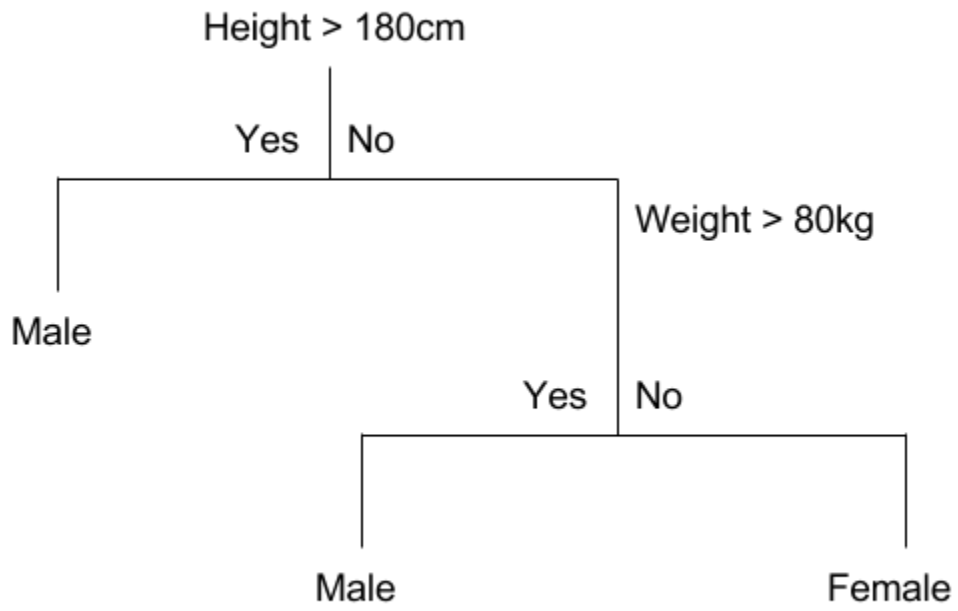
Decision Trees

- Classification and Regression Trees or CART for short is a term introduced by Leo Breiman to refer to Decision Tree algorithms that can be used for classification or regression predictive modeling problems.
- Classically, this algorithm is referred to as “decision trees”, but on some platforms like R they are referred to by the more modern term CART.
- The CART algorithm provides a foundation for important algorithms like bagged decision trees, random forest and boosted decision trees.

CART Model Representation

- The representation for the CART model is a binary tree.
- This is your binary tree from algorithms and data structures, nothing too fancy. Each root node represents a single input variable (x) and a split point on that variable (assuming the variable is numeric).
- The leaf nodes of the tree contain an output variable (y) which is used to make a prediction.
- Given a dataset with two inputs (x) of height in centimeters and weight in kilograms the output of sex as male or female, below is a crude example of a binary decision tree (completely fictitious for demonstration purposes only).

CART Example



CART Example

- With the binary tree representation of the CART model described above, making predictions is relatively straightforward.
- Given a new input, the tree is traversed by evaluating the specific input started at the root node of the tree.
- A learned binary tree is actually a partitioning of the input space. You can think of each input variable as a dimension on a p-dimensional space. The decision tree split this up into rectangles (when p=2 input variables) or some kind of hyper-rectangles with more inputs.
- New data is filtered through the tree and lands in one of the rectangles and the output value for that rectangle is the prediction made by the model. This gives you some feeling for the type of decisions that a CART model is capable of making, e.g. boxy decision boundaries.
- For example, given the input of [height = 160 cm, weight = 65 kg], we would traverse the above tree as follows:

Learning a CART model

- Creating a CART model involves selecting input variables and split points on those variables until a suitable tree is constructed.
- The selection of which input variable to use and the specific split or cut-point is chosen using a greedy algorithm to minimize a cost function. Tree construction ends using a predefined stopping criterion, such as a minimum number of training instances assigned to each leaf node of the tree.

Greedy Splitting

- Creating a binary decision tree is actually a process of dividing up the input space. A greedy approach is used to divide the space called recursive binary splitting.
- This is a numerical procedure where all the values are lined up and different split points are tried and tested using a cost function. The split with the best cost (lowest cost because we minimize cost) is selected.
- All input variables and all possible split points are evaluated and chosen in a greedy manner (e.g. the very best split point is chosen each time).
- For regression predictive modeling problems the cost function that is minimized to choose split points is the sum squared error across all training samples that fall within the rectangle:

$$\text{sum}(y - \text{prediction})^2$$

- Where y is the output for the training sample and prediction is the predicted output for the rectangle.
- For classification the Gini index function is used which provides an indication of how "pure" the leaf nodes are (how mixed the training data assigned to each node is).

$$G = \text{sum}(pk * (1 - pk))$$

- Where G is the Gini index over all classes, pk are the proportion of training instances with class k in the rectangle of interest. A node that has all classes of the same type (perfect class purity) will have G=0, where as a G that has a 50-50 split of classes for a binary classification problem (worst purity) will have a G=0.5.

Greedy Splitting

- For a binary classification problem, this can be re-written as:

$$G = 2 * p1 * p2$$

or

$$G = 1 – (p1^2 + p2^2)$$

- The Gini index calculation for each node is weighted by the total number of instances in the parent node. The Gini score for a chosen split point in a binary classification problem is therefore calculated as follows:

$$G = ((1 – (g1\_1^2 + g1\_2^2)) * (ng1/n)) + ((1 – (g2\_1^2 + g2\_2^2)) * (ng2/n))$$

- Where G is the Gini index for the split point, g1_1 is the proportion of instances in group 1 for class 1, g1_2 for class 2, g2_1 for group 2 and class 1, g2_2 group 2 class 2, ng1 and ng2 are the total number of instances in group 1 and 2 and n are the total number of instances we are trying to group from the parent node.

Stopping Rule

- The recursive binary splitting procedure described above needs to know when to stop splitting as it works its way down the tree with the training data.
- The most common stopping procedure is to use a minimum count on the number of training instances assigned to each leaf node. If the count is less than some minimum then the split is not accepted and the node is taken as a final leaf node.
- The count of training members is tuned to the dataset, e.g. 5 or 10. It defines how specific to the training data the tree will be. Too specific (e.g. a count of 1) and the tree will overfit the training data and likely have poor performance on the test set.

Tree Pruning

- The stopping criterion is important as it strongly influences the performance of your tree. You can use pruning after learning your tree to further lift performance.
- The complexity of a decision tree is defined as the number of splits in the tree. Simpler trees are preferred. They are easy to understand (you can print them out and show them to subject matter experts), and they are less likely to overfit your data.
- The fastest and simplest pruning method is to work through each leaf node in the tree and evaluate the effect of removing it using a hold-out test set. Leaf nodes are removed only if it results in a drop in the overall cost function on the entire test set. You stop removing nodes when no further improvements can be made.
- More sophisticated pruning methods can be used such as cost complexity pruning (also called weakest link pruning) where a learning parameter (alpha) is used to weigh whether nodes can be removed based on the size of the sub-tree.

Exercise

- Try and solve the Leaf exercise using all the classifiers which you have learned

Thank you!


thank you
danke
merci
gracias
obrigado
dziękuję
bedankt
go raibh maith agat
teşekkür ederim
ngiyabonga
شكراً جزيلاً
謝謝
спасибо
рахмат
Баярлалаа
dank je
tapadh leat
mauruuru
hvala
sukriya
kop khun krap
terima kasih
grazie
arigatô
dakujem
мерси
mochchakkeram
faafetai lava
kiitos
dankie
dhanyavad
nanni
nandri
bayarlalaa
gràcie
enkosi
köszönöm
akun dankon
aciù
vinaka
спасибі
blagodaram
misaotra
matondo
paldies
grazzi
mahalo
хвала
asante
manana
obrigada
terki
murakoze
djiere dieuf
tau
дякую
mamnun
chokrane
chnorakaloutioun
gratias ago
gràcies
sulpáy
taiku
takk
trugarez
sobodi
dèkuji
mèsi
sagolun
najis tuke
tanemirt
rahmet
dhanyavadagalu
shukriya
merce
didi madloba
kam sah hamnida
তোমাকে ধন্যবাদ
감사합니다
xiexie
diolch
ευχαριστώ
welalin
tack
barka
mersi
spas