COMP 47460

Machine Learning

Assignment 3

Conor Murphy

20205251

20/12/2020

Introduction

The following analysis was conducted using the cluster_a3.arff file, as the file only contains three features there is no need for any form of feature selection. The data also does not need to be normalised or any data transformation actions taken on it. For this data set the correct clusters are already classified which will allow for comparison between proposed clusters with different algorithms and the actual values. The dataset contains 788 instances which will allows for the creation of large and clear clusters.

Question 1

A)

Weka has multiple different evaluation methods for measuring the quality of a cluster these include

- Sum of squared errors
- Classes to Cluster evaluation
- Visually Comparing

Sum of Squared Errors

This is probably the simplest of the evaluation measures for clustering in Weka, it is the sum of squared difference between each observation and the nearest centroid. This is generally calculated using some distance measure, as the data is only two dimensional and continuous the Euclidean distance measure is suitable as the distance measure. When comparing results, a lower SSE value is desired as it demonstrated good cohesion between the different data points in a cluster

Classes to Cluster Evaluation

This Evaluation method is only possible if the data has preassigned classes, this is not often the case as clustering is an unsupervised form of machine learning and will often not have any predefined classes with the purpose of grouping together similar items in the data. For this analysis, the dataset does include 7 predefined classes allowing for the use of this evaluation measure. The Classes to cluster Evaluation measure essentially will ignore the classes at first, it will then apply the desired clustering algorithm to the data set and compare the clustering results to the predefined correct classifiers. This will allow for us to see what clusters work well and what ones are not being correctly clustered. This method will be examined more closely later in the report.

Visual Comparison

One of the best ways to evaluate the performance of clusters it to simply view them, Weka thankful offers this functionality. By viewing a data set we can simply see the cluster that make sense and the ones that do not, this is especially helpful for non-convex shapes. Take for example the k-Means cluster in Figure 1.1, by simply viewing it we can see that the purple and blue clusters are incorrect and should in fact be different clusters, viewing scatter plots of the cluster can also help in understanding the limitations of different algorithms like the K-means clustering based on distance to a centroid in the shown figure. Overall, visually inspecting clustering outputs is a really powerful evaluation method.

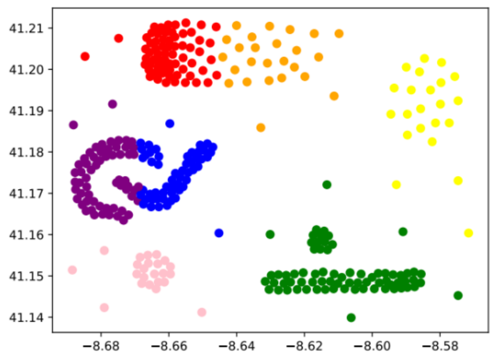


Figure 1.1

B) Before carrying out clustering on the dataset using varying values of K I first wanted to ensure that normalising the data would not yield different results, Using the Normalise function in pre-process I normalised the data and ran it on the new dataset with k=2, as suspected the results were Identical to

those of the standard dataset with the same algorithm, the default settings for the k-Means clustering algorithm were used with some notable parameters being.

Max iterations	500
Seed	10
Distance Function	Euclidean

K	Number of Iterations	SSE	Incorrectly clustered instances	
2	6	83.18709829	385	48.86%
3	18	39.83916779	215	27.28%
4	10	24.76712442	120	15.23%
5	12	21.28237268	82	10.41%
6	10	14.91648811	107	13.58%
7	18	13.63489568	171	21.70%
8	10	12.19149986	246	31.22%
9	9	11.00682843	304	38.58%
10	19	10.2909043	304	38.58%

Table 1.1

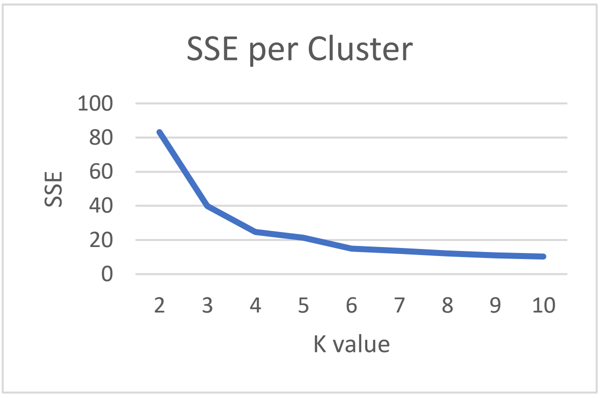


Figure 1.2

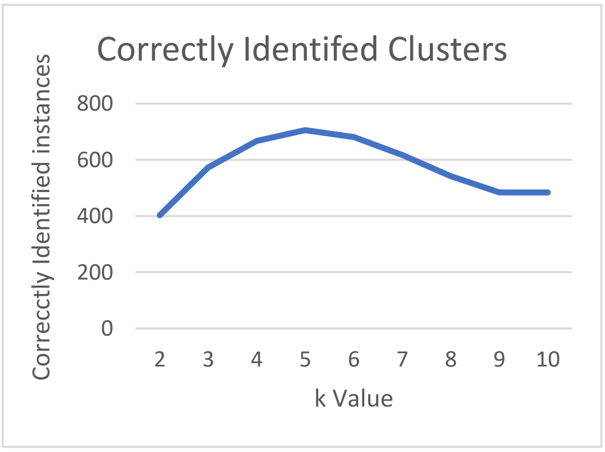


Figure 1.3

Shown above in Table 1.2 is the number of iterations, the Sum of the squared errors and the number/percentage of incorrectly clustered instances for each value of K. Some interesting observations can be made about the data. Shown in Figure 1.2 is the SSE for each different value of K. If we were to only go off of the SSE then the clusters with larger values of K would be better however this makes sense that the value of SSE is consistently decreasing as the distance from the centre of a cuboid would be much smaller as the centroid numbers increase, so as the number of centroids increase the SSE will continue to fall, meaning that the SSE is not a good measure of performance on its own. Shown in Table 1.1 is the total number of correctly classified instance for each value of K, this was generated by subtracting the incorrectly classified classed provided by Weka from the total instances (788). Interestingly when k equalled 5 the algorithm was better than when k equalled 7. This was a shock as prior to the clustering we knew that the data set had 7 clusters, so I expected the optimal number for K to also be 7. The reason why k=5 performed better can be made more apparent when we visually analyse the two images Figure 1.4 & 1.5. In figure 1.5 where k=7 we can see how some clusters (top right) are being separated into smaller clusters where they should not be. Both Graphs highlight a restriction of K-means as they group the bottom left as one whole cluster when in reality it should consist of three separate clusters. This explains why k=7 is not the optimal number of clusters and in fact k=5 is.

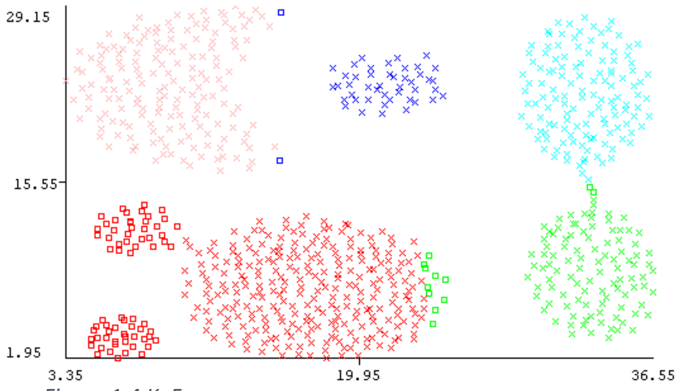


Figure 1.4 K=5

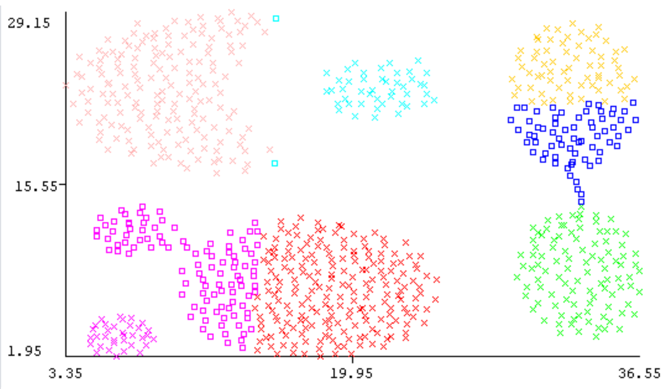


Figure 1.5 K=7

(C) Before carrying out the analysis using the different initialisation methods it is helpful to understand the difference between the random, k-means++ and Farthest First initialisation methods. Random is as the name suggests, choosing completely random points for the initial centroids, Using the seed value in Weka it allows us to reproduce the randomness, however it is often unknow what the best seed to use is and the quality of the clusters differ greatly because of this value. The K-means++ seeks to find clusters that minimise the SSE aka the intra class variance, however this initialisation method still has shortcomings and can perform poorly against other initialisation methods as it focusses on SSE and as shown is table 1.1 SEE does not always mean more accurate clusters. Farthest First will pick a random point and then select the farthest possible point as the next centroid this will ensure centroids are far away from each other at initialisation which should improve separation between clusters. The following are the results from using the three initialisation method.

Initialisation K=7	Num Iterations	SSE	Incorrectly clustered instances	
Random	18	13.6349	171	21.70%
k-means++	15	14.67756	232	29.44%
Farthest First	15	12.80159	110	13.9594

Table 1.2

From the results some interesting observations can be made. K-means++ performed the worst with a nearly 30% incorrect clustered instance rate, it was surprising to see how the optimised version of the k-means algorithm performed worse than the random centroid initialisation, however when we observed the results from part b it is clear that this dataset is not ideal for the k-means algorithm so trying to optimise for it may not be the best Idea and so k-means++ is not a good initialisation method for this problem. The Farthest first initialisation method was the best of the three by far and is compared to the results of Table 1.1 it would place in second, this demonstrates that this data set clusters better when the centroids are farther apart from each other for initialisation. This was an interesting result which made me wonder if the results from the previous part could be improved using the farthest first initialisation method, shown below are the results .The results show that this has even less incorrectly clustered instances than the previous best, and shows how initialisation is a key step for generating better clusters.

Initialisation	Num Iterations	SSE	Incorrectly clustered instances	
Farthest First ,k=5	5	21.27532	77	9.77%

Table 1.3

Part B

Linkage Type	Incorrectly clustered instances	
Single	138	17.51%
Complete	96	12.18%
Average	4	0.51%
Centroid	0	0%

Table 1.4

(A) The different Linkage types were set, and the algorithm run on the data set the results are as shown in table 1.4

The Hierarchical clustering using Single Linkage is by far the worst performing this shows that merging clusters by the smallest pairwise distance is not preferred here. The Complete linkage performers somewhat better than the Single with 96 incorrectly clustered instances and shows the clustering being a slightly better when clusters are merged using the largest pairwise distance. The Average and centroid Linkage types yielded the most surprising results with the Average linkage only having 4 Incorrectly classified instances and the Centroid Linkage being a perfect match. The average linkage performs better than Single and complete as apparently this data set clusters better based on average distances of all of the pairwise items in a cluster. The Centroid Linkage type was a perfect match to the predefined classes which was a genuine surprise and clearly merging based on the distance between centroids is the best option of this dataset.

(B)

	Instances per cluster								
Cluster	Single		Complete		Average		Centroid		Actual Classes
0	168	21%	170	22%	170	22%	170	22%	170
1	1	0%	91	12%	34	4%	34	4%	34
2	1	0%	250	32%	273	35%	273	35%	273
3	307	39%	94	12%	106	13%	102	13%	102
4	232	29%	39	5%	126	16%	130	16%	130
5	45	6%	99	13%	45	6%	45	6%	45
6	34	4%	45	6%	34	4%	34	4%	34

Table 1.5

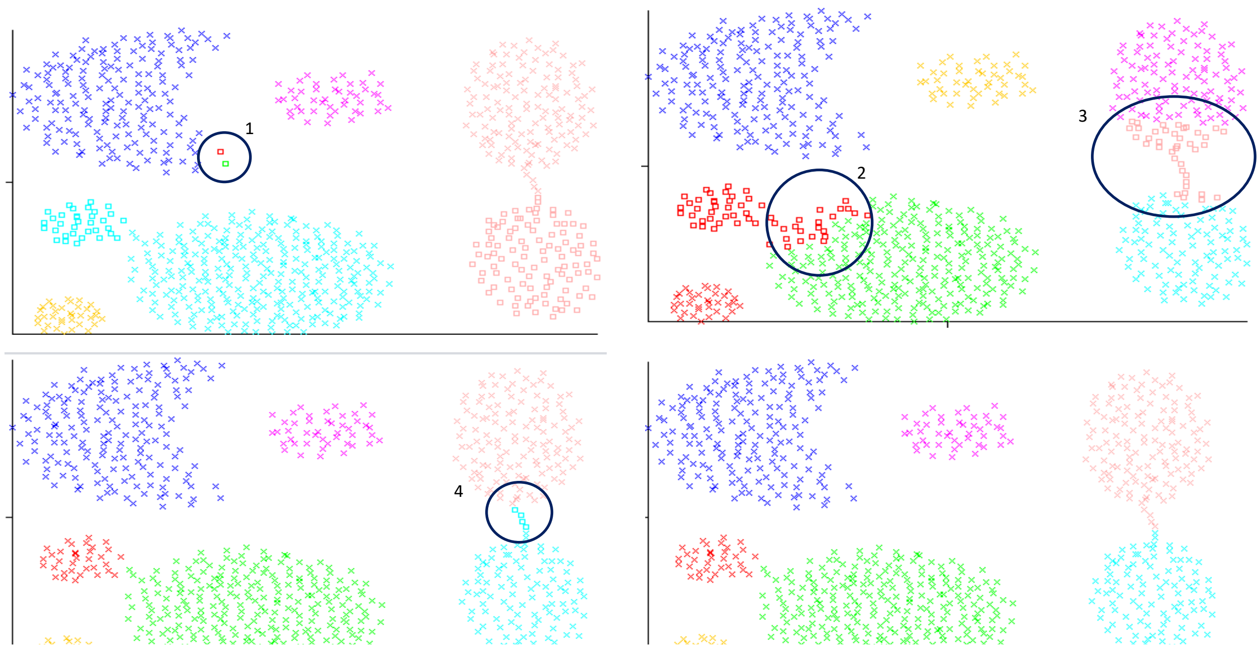


Figure 1.6 Top Left(Single), Top Right(Complete), Bottom Left(Average), Bottom left(Centroid)

Shown in the table above and the graphs, some interesting observation can be made, the most interesting ones are highlighted with numbers assigned.

1) Two clusters with only one item each, these should belong to the blue cluster. This is a result of the closest pairwise distance being used.
2) The red cluster clearly encroaching on the green cluster, a result of merging based on largest pairwise distances
3) A new cluster is formed where it should all be the one Pink cluster
4) 4 entries should belong to the light pink cluster, but is a very minor problem

C) As in Part A the quality of the clusters of k=7 was not even the optimal within all values of K, so it simply cannot compare to the performance of the Hierarchical cluster shown in part B. All linkage types even do a better job and have a significantly lower Incorrect cluster instance rate, even the single linkage hierarchical clustering only has 138 incorrectly clustered instances which is better than the 171 viewable in Table 1.1 for k=7. All of the hierarchical methods perform better than the k-means clustering algorithm with the average and centroid performing considerable better.

Each machine learning problem will be have different optimal clustering algorithms, in this instance the k means algorithm is not the right choice for the problem. The data does not suit the algorithm as it does not handle smaller more densely packed clusters very well, and this is evident from the analysis. On the other hand, the hierarchical clustering especially using the average and centroid linkage types performs significantly better and is definitely the clustering approach that should be used. It is useful to note that this comparison would not have been possible without the classes to cluster comparison and generally it is not this simple to compare the efficiency of two separate clustering methods. I was interested to see how the results compared to other clustering algorithms in particular density based one, I ran the dbscan (Density-based spatial clustering of applications with noise) on the data set and even with an optimised version of that It still had 35 incorrectly classified instance. The centroid initialised Hierarchical clustering is the optimal algorithm for this dataset.

D) As mentioned in Part A visually examining the cluster can be a powerful evaluation method, this can also be said for the tree diagram of the clusters which is also known as a dendrogram. We do however encounter a problem when viewing dendrograms when the data set is extremely large as not much information can be decerned from them, this is certainly the problem I encountered here. Shown in Figure 1.7 is a portion of the tree generated from the centroid initialised hierarchical clustering. This can however be used to help understand the process of cluster assignment. Here as the clustering is agglomerative all of the instances are placed at the base of the diagram and from here based on the proximity of centroids (as it is using centroid initialisation) the clusters that are closest together are merged to form new clusters. This process is then repeated until we meet the threshold, which is 7 in this case, however if there was none the end result would be one cluster consisting of all of the instances. If the algorithm were set to work Divisively then this process would be reversed with all of the instances starting out are one group and slowly dividing them. These diagrams can also serve as a useful guide to see the iterative process of hierarchical clustering as at each stage clusters are being merged based on some linking type.

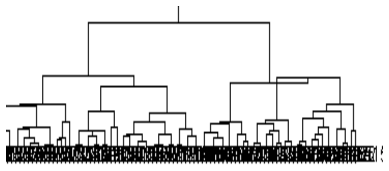


Figure 1.7

In this instance the trees do not offer that much information as there simply is too many nodes, because of this the numbers are not very discernible thus the cluster groping is not clear, A better method to compare these clustering algorithms would be to examine the information displayed in the tables and scatter plots of part B. If there were fewer entries then this would be a different story and more information would be able to be made out from the tree diagrams.

Question 2

Part A

Sometimes in Machine Learning the computation cost of running a certain algorithm on a data set may simply be too large, this is where dimension reduction can help. Dimension reduction can take on two forms the first is Feature selection i.e., chose the best features and only use them for the analysis, the second is Feature transformation, this is where a new smaller and more compact dataset is generated which will retain as much information as possible from the initial dataset.

Feature Selection algorithms can be broken down into two categories wrapper and Filter methods. Filter methods will select features based on some predictiveness feature like information gain, so to evaluate this we can simply run the feature set using a certain Machine learning algorithm like KNN, evaluating the quality of the filter method can be difficult as it will perform differently for different algorithms i.e. the features selected may work better in a decision tree over KNN, so generally testing on different classification algorithms is the best way to judge the quality of the feature selection. For the wrapper method it will generally return the optimal features for a given algorithm like i.e., decision tree, this could be evaluated by comparing it to the filter selection method and seeing how much better it performs compared to the generalised feature selection method.

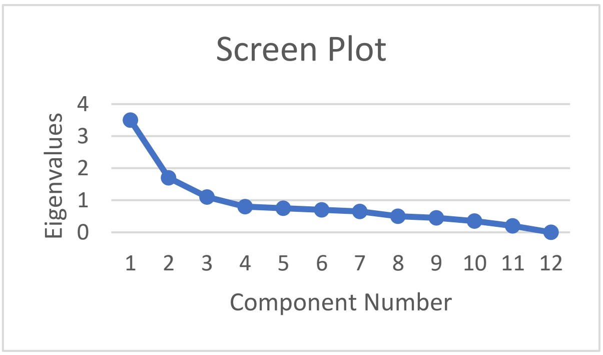
When analysing dimension transformation techniques, it becomes a bit more difficult as we are changing the dataset. One Example of Dimension Transformation is Principal Component Analysis (PCA) this is an unsupervised algorithm which will perform dimension reduction while maintaining as much variance as possible in the dataset. PCA would in theory be used to reduce a dataset with 4 features down to a new data set with 2 features, as 2 dimensional datasets can easily be graphed. Before we use this newly reduced data set, we need some measures to evaluate the efficiency of the dimension reduction.

When we carry out Dimension Transformation the goal is to keep as much information about the originals data distribution, as we fit the data to less dimensions some amount of information will be lost in how the original points are related to each other, the goal is to minimise this loss of information. When performing PCA in Weka we can use the cumulative variance percentage metric to evaluate the dimension reduction. This represents the percentage of the variance covered by each component take the below table for the Iris dataset

Component	Eigenvalue	Proportion	Cumulative
1	2.91082	0.7277	0.7277
2	0.92122	0.23031	0.95801
3	0.14735	0.03684	0.99485

Observe the cumulative column, this represent the total percent of variance covered by each PC so if the goal were to achieve 95% variance then we would need to use the first two components generated but if the goal was to achieve 99% variance then we would adopt the third component also. The proportion represents how much added variance the component adds, the goal with PCA is to have initial larger values of variance in as few components as possible so using the cumulative variance percentage we can evaluate how well the dimension reduction was carried out within our desired variance levels.

Another method to observe the quality of the data reduction technique is a screen plot, this highlights the Eigenvalues on the y axis and the PCA component number on the x axis, this is similar to the elbow method as we can chose the point of the plot where the eigenvalues level off and then this is the number of PCA components that will be used. An example of a screen plot is shown to the right, in this instance the elbow evens out after 3 components are covered so this is the number that should be chosen, this method can help to visualise the effectiveness of PCA as it allows us to see if enough of the variance is covered in the first few PCA components, as this is desired.



Overall, the ability to evaluate quality of dimension reduction is extremely important as the reduced features will be used for subsequent analysis so having confidence that they represent the original dataset well is imperative to be able to confidently use the data for constructing a model.

Part B

Generally, when creating ensembles, the accuracy of the ensemble should be higher than the accuracy of the individual members in the ensemble. However, in some cases the accuracy may not improve, there can be many reasons for this. In this example boosting is being used to for 5 separate models and then these models are being combined in an ensemble. Some problems can arise from boosting as the method focuses on using the misclassified examples to iteratively correct itself, this is generally seen as a major benefit, but it does come with the risk of overfitting the model to the training data. This can result in the unfortunate event that the model is overfit, and it can give too much weight to the noise in the data, this could result in the ensemble of boosting models performing worse as each model may be overfit and thus not very useful for unseen data. This is not very likely to be the cause as even if each model was overfit the diversity should help to compensate for that, but it could be a contributing factor.

Another possible cause for the ensemble of boosting models to not perform better than each individual model is that that the ensemble does not have adequate diversity. If each Boosting model is extremely similar to one and other then there will be no disagreement between them, this is not desired, and it can lead to poor overall performance. This concept of diversity is a major component of ensembles as it is necessary for there to be some level of disagreement between classification models for the overall ensemble to have better accuracy than each of the models that make up the ensemble. In this case the reason that the ensemble does not perform better is very likely because there is this lack of diversity. To rectify this the diversity among the models needs to be improved which will ensure that there is less correlation between the different prediction models, this could possibly be achieved using bagging or a random feature approach as these approaches will attempt to generate independent models that will hopefully have a good level of disagreement between them which in turn will lead to the average prediction being more accurate than the accuracy of the individual classifiers in the ensemble.

Part C

It is often the goal for us to have highly optimised and accurate models, naturally we want the best model possible for machine learning analysis, this is where some evaluation metrics are helpful, one of these methods is a test train split this is where some of the total data set is held out while the remaining data is used to train the model. The test data is then used as unseen data on the model and from this we can observe how the model handles unseen generalised data. Typically, the split is 66% of the data is used for training and 33% for testing, this can also be changed to suit different datasets.

k-fold cross validation, typically 10 folds is another evaluation method. This is where the training data is split into 10 equally sized subsets. One of these subsets are held out and then k-1 are used to train the data. The one subset that was held out is then used on the newly trained model and the results are saved, this process is then repeated on each of the subsets with each of them being used as the test set once. The results are then aggregated, and the overall accuracy of the model is then returned.

Using cross evaluation is extremely helpful as it would help to reduce the chance of overfitting, for example if you only used the test data on the model then the model could start to overfit to this model and as a result return high accuracy for the training data but will not generalise to new unseen data. By using k-fold cross validation it will allow for the testing of unseen held out data and thus the accuracy of the model will be closer to the actual truth, this is also more thorough than just one test train split.

In a lot of the literature the terms Training and validation sets are used interchangeable, but they are in fact different. Validation is similar to the test set as it also held out from the training model, however it is not used to see how effectively the model handles unseen data, instead it is used to tune the parameters of the model which will allow for better performance. The validation set can be used to compare the performance of different parameter inputs on the data to decide which ones yield the best results and thus should be adopted. It is important to note that the data is not tested on the validation set and after the parameters are tuned it is still tested on the held out test set. If it was also tested on the validation set, then this would introduce a level of bias as the data was used to optimise the parameters. As with the test set earlier there are different split ratios the most common are shown below.

Training Set (50%)	Validation Set (20%)	Test Set (30%)

Training Set (40%)	Validation Set (20%)	Test Set (40%)

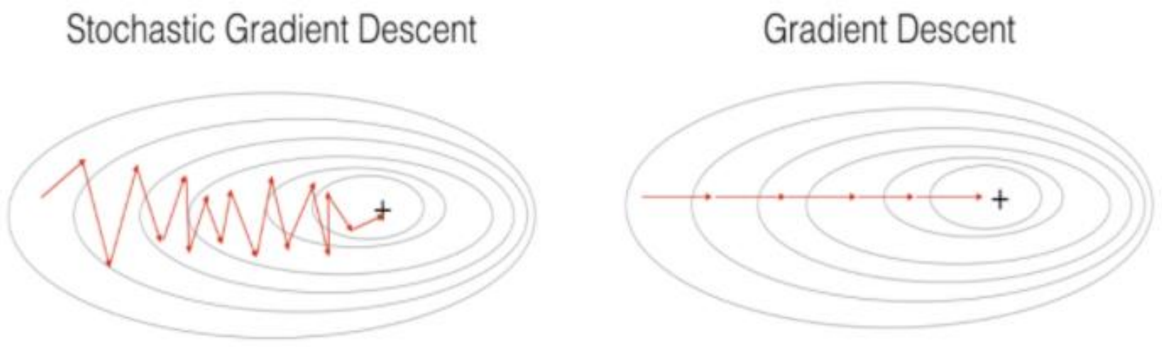
This Three way approach is know as a three way hold out strategy and using it in tandem with k-fold cross validation will allow for the best performing and accurate analysis of models that will do the best job at generalising unseen data. This is the reason why we need to use a validation set alongside the training and test sets.

Part D

In the event that your dataset has high variance this will of course affect which classification model should be used. This very much so speaks to the bias variance trade off in machine learning, Bias is the measure of the difference between the classifiers predicted value and the correct target value, for example if a model is extremely overfit it will have no bias and conversely if it is not that well fit like in linear regression the bias can be rather large. Variance measures the error from sensitivity to small fluctuations in the training set, in this instance our extremely overfit model would have very high variance as it does a poor job of classifying new data, the linear model would encounter the opposite and in fact have a low variance. This problem is know as the Bias variance trade off and it is a major concept in machine learning.

When deciding what classification algorithm to use it is important to understand if they are stable or unstable learners. A stable learner is one that will not be affected by small changes in the input data and an unstable learner will be affected by these changes. This is where the variance comes into play as if different subsets of the data are used for training then this will lead to very different models being produced by unstable learner algorithms, this means that if the variance is high a stable learner may be preferred over an unstable learner. Some examples of Unstable learner algorithms are Decision trees and Neural networks as they can be heavily influenced by changes in the training data. Some examples of stable learners would be Linear regression as mentioned earlier or KNN as they are not affected by the small changes to the training data. In this instance of a dataset with high variance a stable learner algorithm would be preferable over an unstable on, however as is always the case with machine learning the context of the analyse should also be considered, for example if this data set were being used in an ensemble then possibly using an unstable algorithm would help with diversity and could result in an overall classifier that performs very well. This demonstrated how different types of learning algorithms can be used to handle different levels of variance in the training dataset.

Part E



Both Batch Gradient descent (BGD) and Stochastic gradient descent (SDG) have the same goal in mind, to find the minimum of a function, for example the minimal loss function for a logistic regression model. Before discussing the differences between BGD and SGD it is useful to know how each of these algorithms work. In Batch Gradient descent all of the training data is used together, a point on the slope will be found and its gradient calculated, based on what the gradient is it will determine the size of the jump in the opposite direction (ensuring it goes downwards). This step will then be repeated as many times as necessary until either the local minimum is found, or we arrive at value very close to it. Using BGD the steps in the fist few iterations will be large and will narrow as the minimum value is approached i.e., lowest loss function

SDG differs as it uses only one training example that could be at random or some selection criterion can be applied. This one sample is then used to calculate the loss function at this point, which is recorded, a new point is then selected, repeating the process . As this happens it will slowly converge toward the local minimum and it can often find a value very close to the minimum in few steps. The big difference between BGD and SGD is the fact BGD iterated through the entire training set and which can make it quite slow for large datasets as SGD only uses one sample per iteration it is a lot faster and generally better suited to larger datasets. If the desired output is the actual minimum or as close as possible to it then DGD may be preferred as due to its longer computational time will generally find more optimised solution, SGD is still accurate, but it could be expected to just give a value that is sufficiently close to the minimum. One of the other downsides to SGD is the fact that once it reaches close to the minimum it will not terminate but instead generally it will continue bouncing around between values close to the minimum which leads to unnecessary steps, BGD does a better job at stopping closer to the minimum. Another approach could be to possibly use mini-batch gradient descent which splits the entire data set into smaller batches and then calculates the gradient descent of each batch, this approach can take advantage of paralysation and thus will increase the performance.

Overall, both Batch gradient descent and stochastic gradient descent both have their advantages and their disadvantages, one is not necessarily better than the other and as always, the best solution should be chosen in regard to the context of the machine learning analysis and the constrains that matter the most should determine which gradient descent approach to use.