Report

# Clustering

**Report Task 1**

The K-means algorithm separates data by minimising the sum of squared distances between each data point. However, Euclidean distances between data points tend to become uniformly large when dealing with high-dimensional data [1]. To tackle this problem, I reduced the dimensionality of the data using PCA to ensure that only the most relevant features of the data are kept. I used the “mle” parameter which automatically chooses the optimal number of principal components. Additionally, I scaled the data to have unit variance which prevents features with bigger variances from dominating the effects of those with smaller variances. I avoided subtracting the mean as this approach tended to create one big cluster. This likely happens due to the data positioning around the origin losing the meaningful distance between groupings. Then I applied KMeans on the scaled data with random initial centroid positions which tended to produce a more even size of clusters compared to the “k-means++” option in sklearn.

I applied the Gaussian mixture model on the same pre-processed data from K-means since they both rely on distance metrics. Considering the high-dimensional data, I picked a tied covariance structure (all the clusters have the same covariance matrix). Separate matrices in clusters mean more parameters, which could lead to recognising irrelevant variances across dimensions and overfitting.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **K-means** | **Gaussian mixture model** | **Random baseline** |
| **Error count** | 3712128 | 2211682 | 6183659 |
| **Total pairs with the same label** | 7190302 | 7190302 | 7190302 |
| **Accuracy percentage** | 51.62% | 69.25% | 14.29% |

**Report Task 2**

**Report Task 3**

High dimensional data causes Euclidean distances to become less meaningful affecting the performance of the K-means and GMM model, also known as the curse of dimensionality. Additionally, groupings of this data are unlikely to be spherical and of equal variance as K-means assumes, making it a bad fit for this kind of data. The Gaussian Mixture model gives better results due to its ability to capture variations in the data that K-means can’t, but on the other hand, is limited in its accuracy due to the structure of the data and the initial centroids. The random baseline has a very low accuracy which highlights that the k-means and GMM models are capturing meaningful patterns and are a lot better than random guessing.

# Classification

**Report Task 4**

Support vector machines require extensive training time for large datasets like the Covertype dataset. For non-linear SVMs usually the Gram matrix is computed making the training time scale at least quadratically with the number of samples which becomes impractical beyond the thousands of data points [2]. The LinearSVC is a faster implementation of a Support Vector Machine in the case of a linear kernel. It uses the liblinear algorithm which avoids computing the Gram matrix and scales linearly to the number of samples. The disadvantage of this model is the inability to capture non-linear patterns within the data and achieves an accuracy of around 70%.

**Report Task 5**

|  |  |  |
| --- | --- | --- |
| **lbfgs** | 61.23 | NA |
| **liblinear** | 70.48 | 68.01 |
| **newton-cg** | 69.06 | NA |
| **newton-cholesky** | 71.52 | NA |
| **sag** | 67.86 | NA |
| **saga** | 68.40 | 67.60 |
|  | **L2** | **L1** |

Using cross-validation I applied logistic regression on different solvers and different regularization strengths (L1 and L2). The data shows that the newton-cholesky solver gives the best results. I cross-validated different regularization values (C = 0.001, 0.01, 1, 10, and 100) for this solver, but the results were nearly identical. Because this solver doesn’t support the multinomial approach, I tried newton-cg, the best solver that allows this feature, but got worse results of around 70% accuracy. My final model uses newton-cholesky with a regularization value C=1 with 50 iterations as convergence does happen in those 50.

Decision trees require little data preparation, which is why there is no scaling or dimensionality reduction applied to the data. Initially, I ran the DecisionTreeClassifier with its default parameters - without any constraints on the size of the tree and Gini impurity as the function for measuring the quality of the split. The accuracy score was 100% which clearly stated that the model was overfitting the training data. Applying PCA, in attempt to decrease the dimension with low variance gave the same result. To reduce overfitting, I tried multiple maximum depth values using cross-validation as shown in *Figure 1*. The graph shows that the accuracy doesn’t increase substantially for trees bigger than 25 layers. The same approach was used for the criterion but all models showed similar results. The final model with a maximum depth of 25 gives 98% accuracy on the training data which balances the model complexity and accuracy.

Figure 1. The graph illustrates the relationship between the maximum depth of the decision tree and the accuracy of the model on the validation dataset.

Running a single decision tree overfits the data. When considering ensemble methods, boosting methods are used for reducing bias typically when models are underfitting, while bagging models are used to reduce the variance suitable for our decision tree [3]. I chose the random forest classifier which uses the bagging approach. To tune the parameters, I initially used grid search on the following parameters: 'n\_estimators': [50, 100], 'max\_depth': [10, 20, None] and 'max\_features': ['sqrt', 'log2']. The results showed improvements as the maximum depth and features increased, while the results for the number of estimators didn’t differ a lot. I fixed the number of estimators to 50 and used cross-validation for bigger numbers for the remaining parameters. The results for the number of features plateaued at 10 and at 35 for the depth - which are the final parameters.

**Report Task 6**

Logistic regression is a linear model and is unable to capture the nonlinear relationships between features and underfits the data. These complex patterns can be captured by the decision tree leading to a higher accuracy. The accuracy of the tree is further improved by the random forest by reducing overfitting and improving generalization. The relatively low improvement can be due to the estimators outputting similar results and struggling to capture different aspects of the data. Although a more probable reason is that the single tree already performs well, meaning the remaining errors are likely concentrated in dense regions of the data where classes are harder to distinguish.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Logistic Regression** | **Decision Trees** | **Ensemble method** |
| **Accuracy percentage** | 71.55% | 93.04% | 95.80% |

# Regression

**Report Task 7**

A graph with a line

Description automatically generatedWhen we plot the data, we can notice that the data has a polynomial relationship. The two curves indicate that this relationship is of degree 3. To confirm this, I generated polynomial features on the data from degrees 1 to 5 and validated them using cross-validation. The model applied to the polynomial features of degree 3 showed the best results with the lowest mean squared error as shown in *Figure 2*.

Figure 2. Mean Square Error vs Polynomial features degree for linear regression

first fold of 4 KFold

Considering the small size and dimensionality of the data,for training the neural network I chose the lbfgs optimizer [4]. The errors of the initial model with 100 iterations and a learning rate of 0.01 are shown in *Figure 3*. The high learning rate led to steep improvements but also caused the model to crash by overshooting the optimal weights. To acknowledge this, I lowered the learning rate to 0.0001 and increased the number of epochs to 500 for a more accurate and robust model. Considering that the dataset was scaled to a small range I assumed that it reduces the activation of non-linear transformations in the neural network, making it difficult for the model to recognise the polynomial behaviour. That is why I tried different architectures of the neural network on two different kinds of pre-processed data. I did this through a 10-fold cross-validation because of the small size of the data for more accurate results. The first one was given the data as it is, and the other one [] for an input X, so the model can better recognise the polynomial relationships in the data. The results from *Figure 4* show that the model trained on the pre-processed data on average is superior to the other one. The best result showed the architecture with two hidden layers with 100 nodes and 10 nodes. But other architectures can be considered due to relatively similar results that vary because of the non-deterministic nature of neural networks.

Figure 3. Training error over 100 iterations with a learning rate of 0.01



Figure 4. The chart shows the mean squared errors on different architecture types of a neural network on processed and unprocessed data. (100,3) means 2 hidden layers, one with 100 and the other with 3 nodes

Assuming that the data is a polynomial function of degree 3 I defined the model as y = w0 + w1\* + w2\* + w3\*. I set the prior probabilities of N(0, 20) for w0, w1, w2, N(0,10) for w3 as having more confidence in that the function is of degree 3 and a uniform distribution for the sigma. During training I removed the initial samples(burn-in) for a more accurate representation of the distribution. *Figure 10* shows that the prediction done with the mean of the posterior distributions of the parameter, fits the data well.

**Report Task 8**

A graph with a red line

Description automatically generatedA graph with green dots and red lines

Description automatically generated

Figure 5. Training data (green dots), linear regression predictions (red line)



Figure 6. Test data (blue dots), linear regression predictions (red line))



A graph with green and red dots

Description automatically generated

A graph with a red line

Description automatically generated

Figure 8. Test data (blue dots) neural network predictions (red line)



Figure 7. Training data (green dots), neural network predictions (red line)



A graph with a red line

Description automatically generatedA graph with green and red dots

Description automatically generated

Figure 10. Test data (green dots), posterior predictive samples (red line)



Figure 9. Training data (green dots), posterior predictive samples (red line)



**Report Task 9**

|  |  |  |
| --- | --- | --- |
|  | **Linear Regression** | **Neural Network** |
| **Mean squared error** | 75.07 | 160492.01 |

Since the input data is one-dimensional, the Linear Regression can capture the relatively simple relationship. Even though the Neural Network didn’t give bad results, this model is designed to handle complex, high-dimensional patterns because it requires more data to generalize properly. For this type of data, the simplicity and the deterministic nature of the linear regression model make it superior.

# Hidden Markov Model

**Report Task 10**

The transition probabilities of the first model, which is the model that is not given the true probabilities, show certainty in just one or two transitions but struggles to accurately predict the true values. Looking at the emission probabilities it is almost certain about one emission, while in the second model one there is a more balanced distribution. Due to the small dataset the first model is overfitting the data leading to certainty in both transition and emission probabilities, which can be supported by the more balanced distribution in the in the emissions in model 2. Surprisingly the likelihood of the parameters in the first model is higher than of second one. This is most likely because it isn’t constrained by the true transition probabilities and overfits the data. In other words, the first model has more flexibility to overfit the data and is more likely to generate the same. In conclusion even though the likelihood for the model 1 is higher, the model 2 parameters might be closer to the true probabilities of the model.