Ml_Report

1 Introduction

throughout the report I will be using scikit learns default parameters as a baseline that I will be trying to improve. Scikit learns default parameters are recognised to be good for most general cases. I will be using numpy random state(0) throughout my report, this is to allow reproducibilty of the data provided. The random state chosen is arbitrary just chose 0 to simplify things.

2 Clustering

2.1 Methodology

For this section, I am assuming that the class labels are known. However, I do not know the error function at this point, as it comes later in the exercise and requires knowledge of the class labels. Since the number of clusters is given as 7, this saves time in selecting an optimal number.

To train my model, I need a metric to specify the quality of each cluster. I have chosen the Davies-Bouldin index as the evaluation metric. With this metric, I can measure the quality of each model and optimize accordingly.

2.1.1 K-Means

For K-Means, I began by using the default scikit-learn values without scaling or modifying any of the features in the dataset:

Result: 2.120

To reduce the Davies-Bouldin index as much as possible, I took the following steps:

1. **Scaling:** Since K-Means is a distance-dependent model, I scaled the data. However, scaling only affected the first 10 numerical features, as the dataset contains 10 numerical features and 44 categorical features.

Result: 2.120 (unchanged)

2. **Dimensionality Reduction:** Considering the curse of dimensionality, where high dimensions reduce the effectiveness of distance-based models like K-Means, I applied PCA to retain 95% of the variance in the data.

Result: 2.01

3. I experimented with different variance thresholds to find the optimal reduction. The best Davies-Bouldin index was achieved with 92% of the variance. Reducing the variance further, e.g., to 91% or 90%, resulted in less-defined clusters.

Result for 92% PCA: Davies-Bouldin index = 1.738

For K-Means, the initialization used was `k-means++`, which ensures clusters are initialized as far apart as possible.

2.1.2 Gaussian Mixture Model (GMM)

Due to the probabilistic nature of GMMs, the Davies-Bouldin index is not as effective. GMMs

allow clusters to overlap, unlike K-Means, which aims to create well-defined clusters. Therefore, for GMMs, I used the Bayesian Information Criterion (BIC) as the evaluation metric. BIC measures the balance between model fit and complexity, penalizing overly complex models

The goal is to minimize the BIC. I ran a pipeline combining PCA and grid search to find the best parameters. The pipeline applied PCA with varying thresholds, and the grid search explored combinations of covariance type and covariance regularization.

Grid Search Settings:

- Covariance regularization: $[1e^{-4}, 1e^{-6}, 1e^{-8}, 1e^{-10}]$
- Covariance type: `full`, `tied`, `diag`, `spherical`

Best Parameters:

- Regularization: $1e^{-6}$
- Tolerance: 0.001
- Number of initializations: 10
- Covariance type: `full`

Due to the high dimensionality of the data, visualizing the clusters was not feasible. Therefore, assumptions about the cluster shapes could not be inferred in another way

BIC Score: $-2,942,246$

2.2 Analyzing Results

Explanation of Error Count Variation:

The error counts vary across the methods due to differences in their underlying algorithms and assumptions about the data:

1. K-Means: K-Means uses Euclidean distance and attempts to create well-separated, compact clusters. While it performs significantly better than random clustering, its reliance on Euclidean distance struggles in high-dimensional spaces and with categorical features, leading to substantial error counts. Despite this, it manages to detect some structure in the data shown in table 2, yielding a better accuracy than GMM.

2. GMM (Gaussian Mixture Model):** GMMs allow overlapping clusters, which is beneficial in datasets with soft boundaries. This however was a hard clustering task and such did not make full use of GMMs capabilities. it is also viable by looking at the cluster distribution in table 2 that GMM's cluster distribution was much more different than the real distribution.

Overall the error counts in both is quite high and that could be attributed to both the data having too many dimensions and mostly categorical features

Table 2 shows the number of data points in each cluster, ranked from highest to lowest. It appears that K-Means captured the bias in the dataset toward two dominant clusters while gmm did not.

2.2.1 Overall Observations

Due to the high dimensionality of the data and the predominance of categorical features, I don't believe Gmm and k-means clustering are suitable due to the limitations of distance based approaches.

Method	Error Count	Total Pairs	Error Rate	Accuracy
Random Baseline	16,211,868	18,914,680	85.7%	14.3%
K-Means	10,674,120	18,914,680	56.4%	43.6%
GMM	11,442,210	18,914,680	61%	39%

Table 1: Comparison of Clustering Methods

Cluster	1	2	3	4	5	6	7
Real	4905	3630	603	351	295	164	52
Random	1489	1462	1447	1435	1428	1394	1345
K-Means	4409	4109	615	482	215	126	44
GMM	5288	1884	989	793	581	395	70

Table 2: Cluster Distribution Across Methods

3 Classification

3.1 Challenges with Training an SVM

- SVM computational time rises exponentially with the number of data points (n) and increases with the number of featuress.
- The `fetch_covtype` dataset contains 581,012 each with 54 features thus it becomes not feasible to train an svm on it
- SVM excels in binary classification, but `fetch_covtype` has 7 classes. While strategies such as one-vs-one (OvO) and one-vs-rest (OvR) can handle multi-class classification, This increases computational time as you would need to train 7 classifiers for (OVR) or 21 for OvO

3.2 Methodology

3.2.1 Logistic regression

-Since logistic regression uses a gradient based optimisation approach I scaled the data using StandardScaler this only effects the numerical features. -using pipelining again to test pca to tune the hyperparameters combined with 5 fold cross validation. I found that keeping all 54 features consistently gave me marginal improvements of 0.1-0.3% as such I decided not to reduce dimensionality. The regularisation however was inconsistent sometimes making marginal improvements if decreased i.e. by increasing C to 1000 but that did not hold true across different random states as sometimes C at 0.1 produced the same results as such I decided to set it to 1 as it did not really make a difference.

Decision trees

-scaling was not required for decision trees as it is scale invariant

-seeing we are using a large dataset overfitting was less of a concern. -I ran it with the full set of features because due to the imbalance of data the 'variance' each datapoint holds appears to be skewed because of the two dominant classes as such pca may end up removing features that help identify the classes with low datapoints - I mainly used grid search to see what different values of min samples i.e. if pruning the tree would increase its robustness and yield better

cross validation scores. I tried [2, 5, 10], I kept the number of samples low due to some classes not having as much data -pruning however made the results worse consistently by about 2% so I kept the minimum sample split at 2

-I used cross entropy as the criteria to split the tree as most of features are categorical - I tested my assumption by also running gini, cross entropy consistently provided a 0.2-0.4% improvements on 5 fold cross validation runs. overall I found the default sci-kit learn parameters to be best with the only improvement I was able to make was changing the criteria

Random Forest

-given the performance of decision trees, and its advantage of dealing with both numerical and categorical data I decided to go with random Forest for my ensemble method. -I used cross entropy again and from my previous experience with decision trees above. I kept the parameter

-I started of with 100 models and sampling using the 63% bootstrap rule where you keep 37% "out of the bag", the 100 models was chosen just to tune the hyper parameters a bit quicker. I tested different feature sizes i.e how many features does it train every model with. I tried low numbers like the square root of the feature size or mid range like 17 or 20 and large numbers like 46. 46 gave me the best result at around 96.38% mean 5 fold cross validation score while lower feature sizes were at around 95%. I thought maybe bagging which includes all feature sizes would be better than random forest so I tested it out with 54 features but got a lower accuracy score. So I stayed in the 46 features range. I manually selected some numbers and found that 45 gave me the best result. After finding the feature size, I started testing the number of models to see which number would be best overall I found 500 to be good where it didnt take too long but provided best results at 96.89% mean cross validation.

	Accuracy
Logistic regression	72.5%
Decision Tree	94.4%
random-forest	97.0%

Table 3: Accuracy on test set for each model

The difference in results is expected as logistic regression is mainly used for binary classification it is also a linear model that is less suitable for highly dimensional datasets like this one. Decision trees is performing very well unlike logistic regression it does not assume a linear relationships, and is able to capture the non-linear relations ships between the features and target. It is also able to capture interactions between features as splits consider combinations of feature values. Random forest further improved decision trees by allowing it to generalise more and mitigate any overfitting done because of the the imbalanced dataset.

4 Regression

4.1 Linear Regression

I applied a standard scaler to the training set, as regression models are sensitive to feature scaling. The scaler was fitted on the training set and then used to transform the data. To evaluate the suitability of linear regression, I first plotted the data to assess the relationship between the variables. The plot revealed a distinct S-shaped pattern, indicative of a cubic relationship.

To capture this non-linear relationship, I transformed the input feature x into a feature vector of degree 3 (x, x^2, x^3) and used this transformed data in a linear regression model.

This preprocessing step ensured that the model could effectively approximate the underlying cubic function.

4.2 Neural Network Regression

Given the cubic nature of the data, I implemented a neural network where the first step in the forward pass transforms x into a cubic feature vector. This design leverages prior knowledge of the data's origin. Neural network design lacks strict rules regarding the number of neurons or layers, so I employed Bayesian optimization to explore different architectures and hyperparameters.

For example, I ran separate Bayesian optimization processes for networks with two and three hidden layers, allowing the optimizer to adjust the number of neurons in each layer. While ReLU is a commonly used activation function, its susceptibility to the dying ReLU problem and the cubic function's inclusion of negative values prompted me to explore alternative activation functions. Bayesian optimization was also employed to test different activation functions, as a grid search would be computationally prohibitive given the large number of combinations.

The best configuration, yielding a 5-fold cross-validation mean squared error (MSE) of 3442.13, was:

- **Hidden Layer 1:** 104 neurons
- **Hidden Layer 2:** 184 neurons
- **Hidden Layer 3:** 208 neurons
- **Activation Function:** LeakyReLU
- **learning rate:** 0.00318838

4.3 Bayesian Approach

For the Bayesian approach, I leveraged prior knowledge that the noise in the data follows a uniform distribution. Consequently, I used a prior uniform distribution of $[0, 200]$ for the noise. Since the underlying function is cubic, I modelled it as:

$$y = ax^3 + bx^2 + cx + d$$

where a, b, c, and d were sampled from prior normal distributions due to the lack of exact information about the function. Additionally, I introduced a variable e, sampled from a prior uniform distribution [0, 200], to represent the noise.

$$y = ax^3 + bx^2 + cx + d+$$

This Bayesian framework allowed me to incorporate domain knowledge into the regression process and enhance model performance. thus

4.4 figures

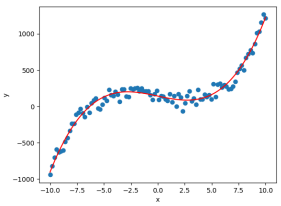

Figure 1: training data fitted with linear regression model

	test data MSE
Linear regresssion	5636
neural network	8156

Table 4: Caption

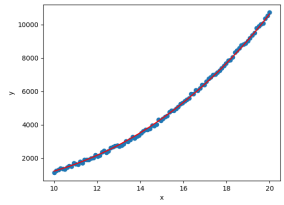

Figure 2: test data fitted with linear regression model

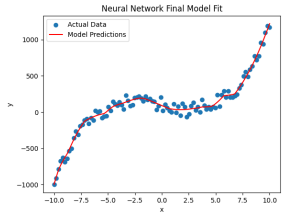

Figure 3: training data fitted with neural network model

4.5 Analysis

Linear regression appears to perform better on the test data than neural network. This suggests that neural network is adding an unneeded level of complexity, The linear regression model appears to approximate the function well but is not able to fully deal with the uniform noise, adding a non linear activation function does not really help the model in fact it made it worse. The difference of performance could also stem from the limited amount of data, neural network require much more training data than a simple model like linear regression needs. Figure 3 shows that neural network may have over fitted to the noise in the training data compared to linear regression in figure 1.

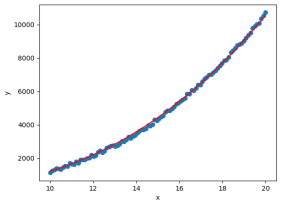

Figure 4: test data fitted with neural network model

5 HMM

Figure 5: the true transition matrix

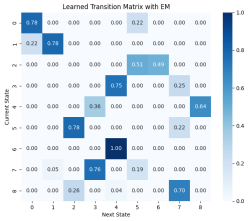

Figure 6: EM learned transition matrix

The HMM that learned the transition matrix through EM deviates significantly from the known true matrix. It tends to assign strong probabilities to one or two next states instead of a balanced distribution among possible next states, this suggests that the HMM converged to a local optimum that does not reflect intended

transitions. This could be because Em's algorithm sensitivity to initialisation with poor initial start and emission probabilities the model may settle for a suboptimal local maximum. In this case the start probabilities learned from the model without the true transition matrix strongly peaked at a single state (2, 1) suggesting that the model believes the process always begins there. When provided with the true transition matrix thus only required to estimate the start and emission probabilities the model again strongly peaked at a single state but at (0,0) instead. I am unconfident in both my start probabilities and emission probability as when I tried to use the ones inferred when the hmm knew the transition matrix, to train a new hmm model to see if it would be able to get close using these fixed start and emission probabilities it did not perform very well and appear to also be a stuck in a local optimum a figure of that is shown below.

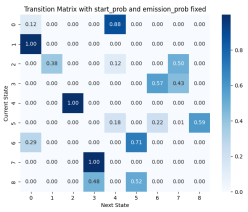

Figure 7: Start_prob and emission_prob taken from hmm with true transition matrix