Machine Learning Report

November 29, 2024

1 Clustering

1.1 Report Task 1 - Justifications

1.1.1 Feature generation

From reading the *covtype.info* file that comes with Covertype, I identified 16 additional features of geological and climatic types that could be calculated from the given soil types. Doing so increased the number of dimensions from 54 to 70.

Initially, the data is transformed using StandardScalar which standardises the data to zero mean and unit variance. This should be done by default, as some machine learning models work best when the data is standardised

After doing feature engineering, the Curse of Dimensionality became a focus. Having 70 dimensions for 10,000 pieces of data can be problematic depending on its structure. To combat this, I applied Principal Component Analysis (PCA) to the data with n_components set to 'mle' (which estimates the optimal number of dimensions) with the objective of balancing keeping the original structure while removing noise. This is method is referred to as MLPCA. 'MLPCA, is that, unlike PMF and other similar approaches, it provides a method to accommodate correlated measurement error'[1]. Covertype is highly likely to have correlated measurement error due to, for example, the instruments used to measure the different features.

1.1.2 Hyperparameter Optimisation

After the appropriate dimensionality adjustments and scaling were applied, the optimal hyperparameters could be selected.

Hyperparameters were selected using skopt's BayesSearchCV to optimise clustering for both KMeans and Gaussian Mixture models (GMM). This method uses cross-validation and probabilistic searching to estimate the hyperparameters with the lowest silhouette score (which is defined by the inter and intra cluster variance as an unsupervised scoring method). The optimal hyperparameters selected for each of methods were:

KMEANS: Algorithm: *lloyd*, Init: *random*, max_iter: *454*, n_init: *23*
GMM: Covariance_type: *diag*, init_params: *random*, max_iter: *120*, n_init: *1*, reg_covar: *0.001*, tol: *0.0001*

1.2 Final Model Performance

Model	Error Count	Total Pairs	Error Rate (%)
Random Baseline	16,204,045	18,906,728	85.7
KMeans	13,191,142	18,906,728	69.8
GMM	7,549,382	18,906,728	39.9

Table 1: Comparison of clustering models with 7 clusters and 10,000 data points

1.3 Report Task 3 - Analysis

Random: Representing the the error of randomly assigning classes. Expected to be higher than actual clustering algorithms. Equivalent to 6/7 error rate, meaning it is correctly working as it has a 1/k chance of correctly assigning cluster **KMeans** has an error rate slightly better than random with 69.8% of pairs in class labels were not clustered together. **GMM** has a significantly lower error rate than KMeans of 39.9%. This means that of the total number of pairs assigned to the same cluster, the GMM clustered 60.1% correctly.

I can conclude that the shape of the clusters can be more accurately modelled by a combination of Gaussian distributions defined by GMMs. The optimum covariance type selected was diagonal ('Each component has its own diagonal covariance matrix, permits components to have different variances along each dimension but assumes no correlation between dimensions.' [2]). This is expected because MLPCA reduces correlation between features.

KMeans models will not fit diagonally-distributed data well, because it creates spherical clusters (assigns based on closest centroid).

Additionally, if we wanted to run unsupervised clustering on this dataset, while not removing most of the dimensions with PCA, we would have to increase the size of the sample subset. Otherwise, the curse of dimensionality limits our maximum performance. The optimum number of dimensions increases with the size of training data.

The usage of these clustering methods has resulted in fairly low accuracies, therefore it isn't a very suitable approach for grouping this dataset. Additionally, clustering is an unsupervised machine learning method and depends heavily on the intrinsic structure of the data.

2 Classification

Classification is a more suitable method with the covertype data set. The covertype dataset gives us the direct class labels, so it is possible to run a supervised model. Then if there are any datapoints without class labels, we can just predict them using our model.

2.1 Report Task 4 - SVM Problem

When I try to train an Support Vector Machine (SVM) classifier on the data, it never finishes training and runs out of memory to allocate. This stems from the fact that the covertype data set has 581,012 rows. If 80% used for training, it gives us 464,810 datapoints. This means that the Gram Matrix will be of size 464,810 x 464,810. This means it will contain 216,048,336,100 elements. All of the 54 features are 32-bit integers [3]. Multiplying these two together, give us 804.8 GB of memory required to compute. This is unfeasible. This is also equivalent to an $O(n^2)$ memory and runtime complexity.

2.2 Report Task 5 - Justifications

All the data was scaled using standard scalar in order to make the data in the form of a standard normal distribution

Cross-validation was used for all the models, so their performance could be evaluated on a validation set without affecting the model training sample size.

2.2.1 Ensemble Methods

Random Forest:

Multiple ensemble methods were trained and the best performing ones on the validation set were passed in as estimators for a stacking classifier. Decision Tree: Pruning (cpp_alpha) of 0.01, 0.1, and 0 were in the parameter space. Optimal parameters for the random forest and decision models were estimated using a Bayes Search.

Decision Tree Best Parameters: ccp_alpha: 0.0, max_depth: None, max_features: 23, min_samples_leaf: 1, min_samples_split: 2, splitter: 'best'. cpp_alpha refers to pruning, so the final decision tree did not prune any of the nodes.

Random Forest Best Parameters: bootstrap: False, criterion: 'gini', max_depth: None, max_features: 30, min_samples$_leaf$: 5, $min\_samples\_split : 6, n\_estimators : 112$

2.3 Report Task 6 - Model Comparison

Model	Val Score	Test Score
Random Forest	0.9361	0.9426
Bagging	0.9383	0.9614
Logistic Regression	0.7245	0.7236
Decision Tree	0.9039	0.9119
Adaboost	0.6090	0.6042

Combining randomForest, decisionTree, and bagging with stacking Stacking test result: 0.963. The stacking predictor has high computational demands due to combining multiple tree-based models. The stacking model has the highest accuracy because it combines the best models in an ensemble method. The decision tree had a high test score due to their strengths at being able to model non-linear relationships in data by splitting into sub-regions. This means it will work for the covertype dataset specifically due to the complex nature of its features. Whereas, logistic regression cannot model non-learn relationships in data, so it's expected that in a complex dataset it will perform badly.

3 Regression

3.1 Report Task 7 - Justifications

Given the data that the function was unknown, I had to try different approaches to find the function.

Linear Regression: I created a method that iterates through 1-5 polynomial degree, and then used

a Polynomial transformation which applies a feature function to convert the data from the polynomial into a straight line.

The model with the Mean Square Error (MSE) that was significantly lower than the rest was with $i = 4$ (cubic function), so I assumed for the linear regression stage that the highest order polynomial in $f(x)$ was 3.

Code taken and adapted from Pytorch quickstart tutorial

Neural Network: The output from the linear regression stage informed my choice of neural network - a cubic performed best - this means the polynomial degree with the highest influence on the results must be three. I decided to explicitly state that the function must be of cubic form in the forward function of the network. I achieved this by applying a feature mapping to the input x to a higher-dimensional space.

This allowed me to then use the neural network to learn the coefficients of each of the basis functions. I checked the validation loss, adding an extra polynomial basis function of degrees up seven. However, the one with the best validation loss had the max degree as three. So the final architecture of the NN must be that of a cubic with less impactful lower-order polynomials.

Due to the fact that the architecture of the forward pass was explicitly stating the function I was trying to model, the network stack only needed one layer with 4 input features (constant, x, x^2, x^3) and one output feature.

The validation data set was chosen to be the rightmost thirty elements of the training set (while keeping them in training set). This was done so I optimise the loss on the right section which is the part of the cubic which is just before the test set (so it is more important to have a lower loss at this part).

I found that the optimiser 'Adam' had the lowest validation MSE over 'SGD', so I used it instead. The neural network implements Xavier weight initialisation for better convergence. It 'initialize[s] the weights such that the variance of the activations are the same across every layer. This constant variance helps prevent the gradient from exploding or vanishing'[4].

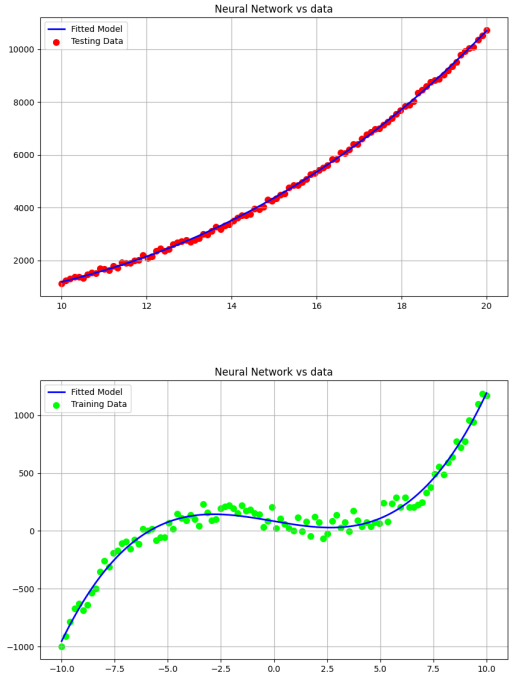
The find_nn_features() function performs empirical testing by:

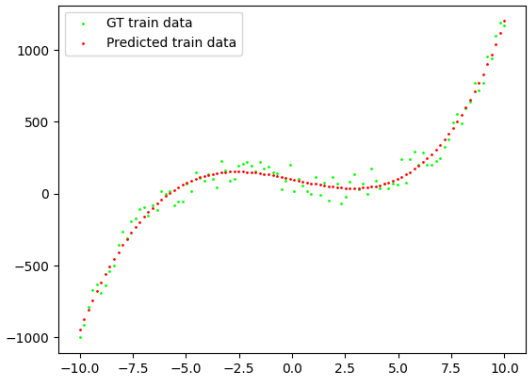
- Implementing early stopping in training: After iterating through 200 sequential epochs without significant improvement, the training will stop. To reduce overfitting beyond this, I subtracted 200 from the total epochs to get the last epoch that improved the validation loss.

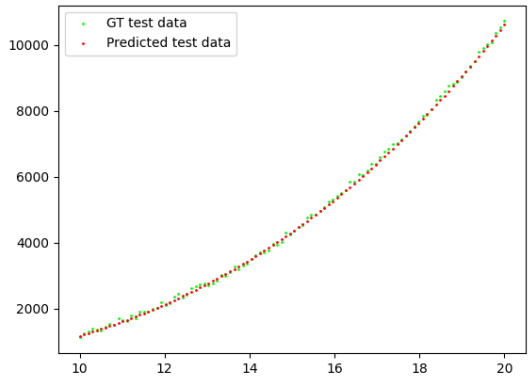
Bayesian Classifier: I defined the distributions in the Bayesian classifier to be 'constant', x, x^2, x^3, and a final 'noise' variable. The advantage of a Bayesian method over the neural network and linear regression I implemented, it that it can also model the uniform noise defined in the task (ϵ-U[0,200]) as part of the function. I did this by passing the noise into the likelihood calculation in the parameter 'sigma'. The classifier sample has 1000 draws, and I run five chains and average the result. This method accounts for the stochastic behaviour of predicting the distributions.

3.2 Report Task 8 - Predictions and Scores

Linear Regression: Test MSE: 5636, Train MSE: 3721, Validation MSE: 3740
Neural Network: Test MSE = 4317, Train MSE = 3764, Validation MSE = 2422







3.3 Report Task 9 - Model difference Analysis

Due to the architecture of my neural network, it will always be trying to fit a cubic function. The test MSE for the linear regression model was 5636, higher than that of the neural network, which had 4317. This means that the linear regression overfits the data more than the neural network. The neural network architecture I implemented used early stopping to reduce the number of epochs, for the purpose of preventing overfitting. Evidently, this method was successful as it now performs better on unseen data than the linear regression model

4 Hidden Markov Models

Each of the Markov models were run with a 1000 iterations. Expectation Maximisation will calculate the posterior probabilities of being in each state, and then update the probabilities to maximise their log-likelihoods.

4.1 Report Task 10 - Result Analysis

TRANSITIONS:
Transition matrix without known transitions:

$$\begin{bmatrix} 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 1.000 & 0.000 \\ 0.282 & 0.000 & 0.000 & 0.000 & 0.000 & 0.015 & 0.703 & 0.000 & 0.000 \\ 0.167 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.833 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 1.000 \\ 1.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.274 & 0.117 & 0.000 & 0.244 & 0.000 & 0.000 & 0.365 & 0.000 \\ 0.000 & 0.000 & 0.000 & 1.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.687 & 0.313 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.617 & 0.000 & 0.383 & 0.000 & 0.000 & 0.000 & 0.000 \end{bmatrix} \quad (1)$$

The diagonal values in this are zero. This shows that the Hidden Markov Model has learned the behaviour that the robot cannot remain in the same position between two adjacent states (has to move). The totals for each row is one, which is as expected. It is beginning to show some semblance to the real transition matrix. But, the probabilities aren't 1/N in the true model. For example, the third row has values 0.167 and 0.833, where it'd be expected to be 0.5 and 0.5. If the rewards file was larger, then it may converge to the true transition probabilities. The order of the states when the neighbours are not defined are random. So state 1 in this model could correspond to any position on the grid.
Transition matrix with known transitions:

$$\begin{bmatrix} 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0.25 & 0 & 0.25 & 0 & 0.25 & 0 & 0.25 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 \end{bmatrix} \quad (2)$$

This is just the transition matrix as defined in the task.

The emission probabilities represent the probability of each reward on each state. For each position, we have the probability that it's reward is assigned either a 0, 1, or 2.

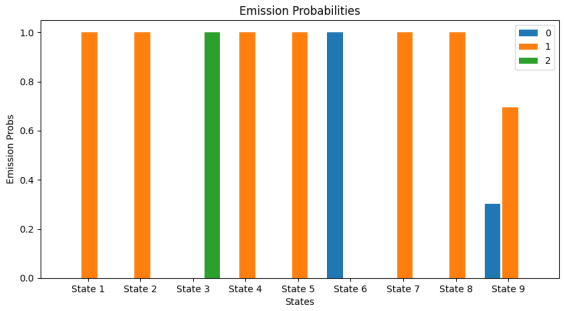


Figure 5 Emission probabilities with random transition matrix

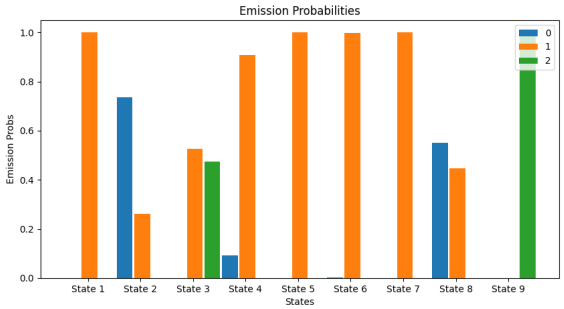


Figure 6 Emission probabilities with given transition matrix

Both of these emission probabilities show a similar pattern the dominant reward is 1.

It is expected that the true transition HMM would more accurately be able to model the transition probabilities, so we can assume that the probabilities shouldn't be exactly at 1.0 as they are for task 14. Instead, I expect that with more reward data, we would see both models converge to emission probabilities similar to Figure 6.

Without transition matrix:

$$\begin{bmatrix} 1.000 \\ 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \end{bmatrix}$$

With transition matrix:

$$\begin{bmatrix} 0.000 \\ 0.000 \\ 0.000 \\ 1.000 \\ 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \end{bmatrix}$$

Both of the starting probabilities have converged to one state. We can tell that this is state 4 (the middle left node) by examining the result from code task 15, since the states are in the correct order.

References

[1]

Wentzell, Peter; Lohnes, Mitchell. *Maximum likelihood principal component analysis with correlated measurement errors: theoretical and practical considerations.* Published: 18 Jan, 1999. https://doi.org/10.1016/S0169-7439(98)00090-2.

[2]

GeeksforGeeks. *Gaussian Mixture Models (GMM) Covariances in Scikit Learn.* Last Updated: 05 Feb, 2024. https://www.geeksforgeeks.org/gaussian-mixture-models-gmm-covariances-in-scikit-learn/.

[3]

Covertype. *Covertype - UCI Machine Learning Repository.* Donated on: 31, Jul 1998. https://doi.org/10.24432/C50K5N.

[4]

Ng, Andrew; Katanforoosh,Kian. *Xavier Initialization and Regularization.* https://cs230.stanford.edu/section/4/