Analysis of Machine Learning Methods on Fashion-MNIST and California Housing Datasets

Lucy Randewich

Abstract

This report will demonstrate understanding of machine learning models and concepts covered in COMS30069 by analysing experiments on Fashion-MNIST[13] and California Housing[9] across the four topics outlined in the coursework brief. Each section outlines methods using equations and/or algorithms as appropriate, before analysing graphs and plots of results gathered and remarking on their implications. After marking, code will be made public at `https://github.com/lucy-randewich/ML_coursework`.

1 Analysing fashion-MNIST

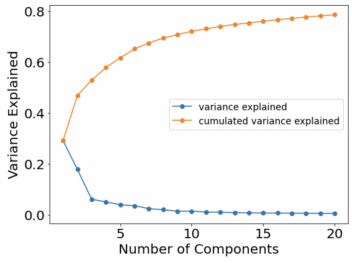
Unsupervised learning techniques can be used to gain intuition into the structure of the Fashion-MNIST dataset, which consists of 60,000 training examples and 10,000 test examples, each associated with one of 10 class labels. Since each example is a 28×28 grayscale image, the dimensionality D of each example is 784.

1.1 PCA

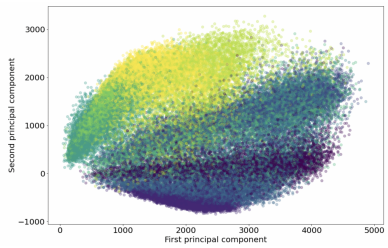
The curse of dimensionality is a term coined by Richard E. Bellman [1] to describe the explosive effect of scaling machine learning methods up to high dimensions in terms of runtime and space. Principle Component Analysis (PCA) limits this effect by projecting data into fewer dimensions while maximising information retained. It is intuitive that removing features with zero variance has no impact as they afford no information about data, so it follows that large optimizations can be gained by keeping only features which explain the highest variance of the data. PCA first calculates the principle components (defined as the eigenvectors of the covariance matrix ordered by eigenvalue) then represents each datapoint x_n for $x \in \{1, ..., D\}$ as a linear combination of these principle components.

$$x_n = \sum_{i=1}^{D} (x_n^T u_i) u_i$$

This splits the data into D dimensions ordered by variance contribution, allowing a value of $M < D$ to be chosen where only the first M dimensions are kept, where M is decided using a scree plot (Figure 1a). The variances explained by the first and second principle components are 0.290 and 0.178 respectively, and so the cumulative variance explained by the first two principle components is 0.468. The diminishing gradient of the plot for subsequent components shows that they do not afford as much information about the data. Figure 1b shows the data having been projected onto



(a) PCA scree plot



(b) Projection onto first two principle components

Figure 1

the first two principle components, each coloured according to its true label. The 2D visualisation shows elliptic trends in the data seen in the clusters of colour, which is a typical use case of a Gaussian Mixture Model (GMM) as each class can be well represented by a multivariate Gaussian.

1.2 Gaussian Mixture Modelling

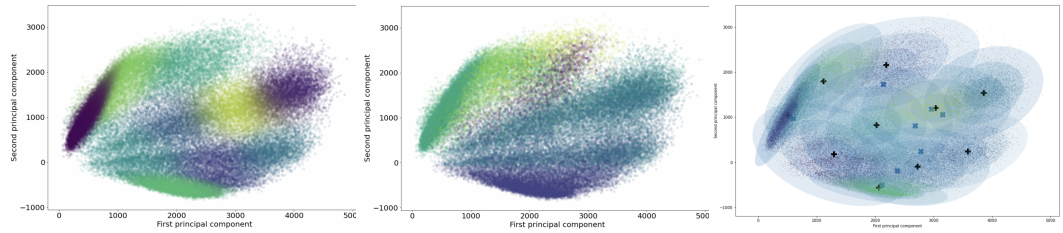
Given datapoints X from k Gaussians, GMM aims to find the Gaussian parameters (μ_k, Σ_k) and mixing coefficients π_k to form a good representation of the data.

$$p(x) = \sum_{k=1}^{K} \mathcal{N}(X|\mu_k, \Sigma_k)$$

This can be combined with a log-likelihood estimation to perform soft clustering, such that each point is assigned a probability of belonging to each class:

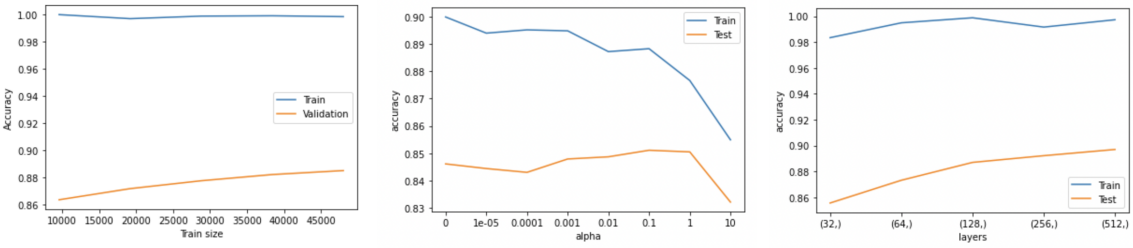
$$\ln p(X|\pi, \mu, \Sigma) = \sum_{n=1}^{N} \ln\{\sum_{k=1}^{K} \mathcal{N}(X|\mu_k, \Sigma_k)\}$$

Figure 2a is the result of applying the GMM algorithm to data projected onto the first two principle components, where colours are calculated using the probability of each point belonging to each class to visualise the soft clustering. Comparing to Figure 1b shows some success, but there are some trends which the GMM fails to represent. Figure 2c visualises the means and covariances of Gaussians produced; some modelled and true means are close together but others differ quite substantially. The PCA process has limited the model's capability since 53.2% of the variance was lost, which clearly contained information which could have improved performance. Classes where the means match closely to true means such as the class at the very bottom of the plots correlate to the classes for which most of the variance is explained by the first two principle components, hence why these points appeared clustered in Figure 1b. Figure 2b shows the GMM algorithm applied to the entire dataset; clusters are more elliptic in shape and show more similarity to those of the true labels in Figure 1b, solidifying the notion that some useful information is lost in PCA. There is a time/accuracy trade-off at play, since the initial model was produced in 1.97s while using the entire dataset saw a tremendous increase of 699s. The dimensionality reduction process also raises interesting ethical concerns due to some classes being favoured as certain projected spaces are chosen over others. For example, a model predicting credit scores could favour a certain ethnic



(a) GMM of Projected Data (b) GMM of entire dataset (c) means and covariances

Figure 2



(a) Learning curves for ANN (b) Regularisation tuning for ANN (c) Hidden layer tuning for ANN

Figure 3

group as a result of changing the feature space through PCA. Attempts to minimise this unfairness is an open research topic, with different fairness methods being considered [7].

2 Classifiers

Classification is a supervised learning task with the aim of assigning each datapoint to one of N classes, where methods seek to find decision boundaries to separate each class. This section analyses classification methods on the Fasion-MNIST dataset.

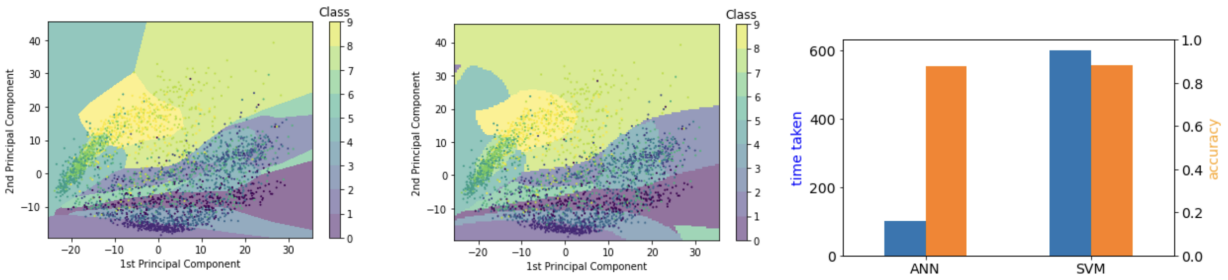
2.1 Artificial Neural Networks

An Artificial Neural Network (ANN) uses interconnected nodes in a layered structure to form an adaptive network of weight vectors. The backpropagation algorithm updates weights in each layer according to the derivative of a chosen loss function (a measure of how close the model got to predicting the true labels). Scikit-learn's default MLP (Multi-Layer Perceptron) yields accuracies of 0.889 and 0.849 on the training and test set respectively. Figure 3a shows training and validation learning curves, which doesn't point heavily towards overfitting as there is no great disparity between the training and validation accuracies (note the small increments of the y-axis). Regardless, there is room for improvement as neural networks have an abundance of settings for hyperparameter tuning. Recent years have seen a resurgence of research on hyperparameter optimization with methods ranging from differentiable architecture searches[6] to modern iterations on the older so-called BlackBox parameter search methods[3].

Figure 3b and c show performance of the model on training and test sets as a function of different

	lbfgs	sgd	adam
Train	0.969	0.964	0.993
Test	0.877	0.884	0.885

Table 1



(a) Decision boundaries of ANN (b) Decision boundaries of SVM (c) Classifier comparison

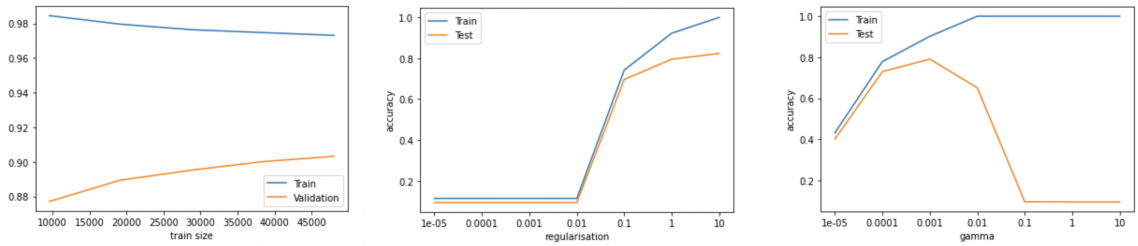
Figure 4

parameters. Regularisation α introduces an extra term to the loss function to manage overfitting. Increasing α encourages small weights, resulting in less curved decision boundaries as the model fits less strictly, while a value of α too large favours large weights, resulting in high bias giving more complex decision boundaries as shown in the downward trend of the graph as α increases from 0.1. This parameter is not particularly sensitive providing the value chosen lies within a sensible range, as all models using αs between 0 and 1 perform to within ± 0.05 of each other. From 3(c), the test accuracy of the model is somewhat directly proportional to the number of hidden layer nodes. However, a larger number of hidden layer nodes results in a vast increase in training time as the model has far more computational steps to calculate weights for each layer. The problem which the model is solving is complex due to the high dimensionality of training examples, so adding more nodes allows it to learn the complex trends. If the dataset were less complex, overfitting would occur when increasing hidden layer size. Table 1 shows that in this case changing the solver has a marginal effect on performance since all led to convergence with a test accuracy range of only 0.008.

Plotting decision boundaries by projecting data onto a 2D space using PCA is a useful way to visualise classification rules (Figure 4a). A subsection of datapoints are scattered on top of the boundaries as a way to validate model success. The complexity of the boundaries shown correlates to complexity of the model; exessively flexuous boundaries are a clear indicator of overfitting, which doesn't seem to be the case here. The plot highlights classes which are often misclassified by the model, but in this case it seems that a high proportion of points plotted match the colour of the decision boundary they lie within, pointing further towards model success.

2.2 Support Vector Machines

The objective of a support vector machine (SVM) is to find an N-dimensional hyperplane separating the data. This is achieved through maximising the margin of the classifier using support vectors,



(a) SVM learning curves (b) Changing SVM regularisation (c) Changing RBF gamma

Figure 5

defined as the points closest to the hyperplane which influence its location. The kernel function defines a degree of similarity between two arguments, which is used to classify new points: the prediction for datapoint x, $y(x)$ is a linear function of the similarities between x and each element of the training data. To avoid keeping the entire training set to make predictions, the 'Kernel trick' keeps only xs where $a_i \neq 0$, where a is defined as the dual parameter such that each prediction is:

$$y(x) = a^T \begin{pmatrix} k(x_1, x) \\ k(x_2, x) \\ \vdots \\ k(x_N, x) \end{pmatrix}$$

Figure 5a shows training and validation curves for Scikit-learn's default setting SVM implementation applied to Fashion-MNIST. The disparity between training and validation accuracies is a clear sign of overfitting. In principle, SVMs are resistant to overfitting as they don't depend on the dimensionality of the feature space, but in practice tuning SVM parameters is fundamental yet non-trivial; the choice of kernel and regularisation parameter are pivotal. Figure 5b produced from a subset of training data highlights the significance of a sensible regularisation value. In this case, only values between 0.01 and 0.1 are acceptable, as the model does not learn sufficiently for values below 0.01 and begins to overfit for values above 0.1.

The model's accuracy metric of 0.900 attained from the test set shows its ability to scale well to unseen datapoints and that it did not overfit to the examples presented at training time. Plotting decision boundaries gives further insight into how well the model performs; comparing Figures 4a and b shows the similarities between the models while also suggesting slight overfitting of the SVM, seen particularly through the curvature of the yellow decision boundary. The kernel used thus far is the radial basis function (RBF) kernel, defined as:

$$F(x, x') = exp(-\gamma||x - x'||^2)$$

γ corresponds to decision boundary spread, so for small γ the curve of the boundaries is low which leads to broad boundaries. This is confirmed by Figure 5c, as the model overfits drastically for larger values of gamma as the function of the decision boundary becomes too complex to be able to generalise to unseen data. Table 2 gives results of using different kernels provided by Scikit-learn, which shows that the RBF kernel is best suited to the Fashion-MNIST dataset. Linear kernels tend to have lower runtimes and perform well on linear data, but they are limited in that they don't have the flexibility of the RBF kernel which Fashion-MNIST requires.

	Linear	Poly	RBF	Sigmoid
Train	0.619	0.636	0.874	0.695
Test	0.597	0.627	0.843	0.686

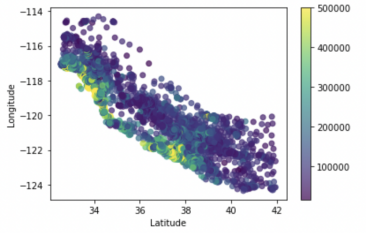
Table 2

Figure 4c compares train times and accuracies of the classification models. The SVM generally performed better than the ANN, with an upper bound test score of 0.884 compared to 0.879 of the ANN. Since both models are sensitive to some parameters, tuning one model more rigorously could well result in that method performing better. In industry settings where models are required to perform inference on terabytes of data in real-time, models are optimised at a granular level to allow for the speed/accuracy trade-off. Training time of the ANN was significantly faster at 100.71s compared to 600.15s of the SVM. Complexity of the SVM is generally considered to be between $O(n^2)$ and $O(n^3)$[11], compared to $O(n)$ for the ANN[8], where n is number of training examples. It can therefore be proposed that ANNs are preferable in situations where time is crucial, while it could be possible to get more accurate results with an SVM. The No Free Lunch theorem[12] reminds us that all models are wrong, but some models are useful; there is no universally best model.

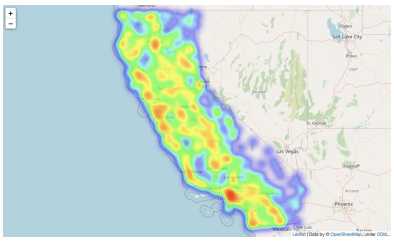
3 Bayesian Linear Regression

This section explores the California Housing dataset before calculating posteriors over each predictor using Bayesian linear regression.

Plotting latitude and longitude values (Figure 6a) shows geographical housing value distribution, where colours correspond to median house price. Plotting values on a heat map (Figure 6b) from the Folium[10] package further highlights the geographical distribution. These plots show that areas of high mean house price are near the coast, so it would be possible to perform dimensionality reduction by creating a 'distance from the sea' predictor, allowing latitude and longitude predictors to be removed without losing much information.



(a) Effect of location on median house price



(b) Heat map

Figure 6

As with any real-world data, preprocessing is required. A good first step is to check for null values, of which this dataset has none. If this weren't the case, the process of imputation[5] could be used to predict missing values using a linear regression with a feature which has a linear relationship

with the feature containing nulls. Of the 20640 rows in the dataset, 846 have values which lie outside a z-score of 3. Removing these makes the model less likely to learn outlier trends which wouldn't generalise for new data. Standardisation is not strictly necessary in Bayesian methods since it does not affect parameter estimates and makes interpreting posterior plots less intuitive, however it does allow chains to converge far more efficiently; scaling reduced runtime drastically from ≈ 140mins to $\approx$ 2mins. Checking for target value skew shows an abnormally large number of records of value 5 due to a ceiling function, and removing these examples gives a more realistic representation of the data.

Bayesian linear regression is a conditional modelling technique to approximate posterior distributions for the parameters of the distribution of the regressand (the dependent variable of the regresssion), enabling tractable sampling conditioned on observed values. PyMC uses the 'No U-Turn Sampler' (NUTS), which is a dynamic version of the Metropolis-Hastings algorithm (algorithm 1). Rather than the binary 'accept or reject' decision in Metropolis-Hastings, NUTS dynamically chooses the next point of the chain through sampling [4].

Algorithm 1: Metropolis-Hastings Algorithm

1: Define single transition probability distribution for a homogeneous Markov chain $z^{(1)}...z^{(i)}$
2: Let the current state be $z^{(\tau)}$, then:
   a) Generate a value z^* by sampling from a proposal distribution $q(z|z^{(\tau)})$
   b) Accept or reject the proposed state based on an acceptance probability
3: If the Markov chain converges to the target distribution then it stays there

Setting prior distributions for all predictors as $X \sim \mathcal{N}(0,\, 20)$ PyMC yields approximate posterior distributions in Figure 7, and Figure 8a gives a summary with means and standard deviations of said distributions. Since the data is scaled, model performance isn't immediately obvious as the approximated means cannot be compared directly to the true means. However, model success can be interpreted by visualisation of true distributions against those produced by the model (Figure 8b). The posteriors replicate the trends of the data to some extent, even more so after the outliers were removed. A key difference is that true datapoints are ≥ 0, so when sampling from the model a floor function of zero must be applied to the sampled points. The run summary in Figure 8 shows that all $\hat{r}$ scores are close to 1, which is another indicator that the run has succeeded as the chains have converged. Since Bayesian linear regression models update with each new datapoint encountered, if a sample of fewer datapoints is used then the model will produce distributions which do not take into account the overall trend of the data, since there is not enough information to allow the Markov chains to converge. Therefore, the posteriors produced by runs which use 50 and 500 samples rather than the full dataset are more 'squiggly' as they have adapted only to the datapoints they have seen. Furthermore, the $\hat{r}$ scores resulting from 50 and 500 samples stray further from 1, evidencing that the Markov chains did not converge.

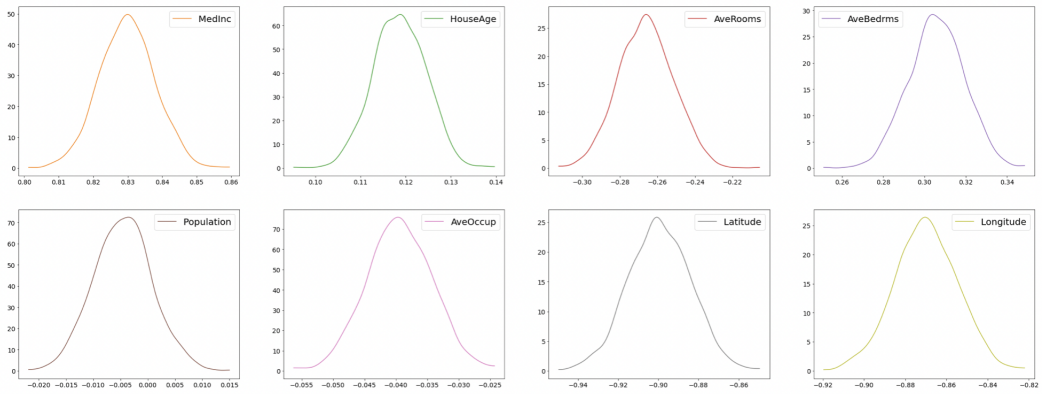
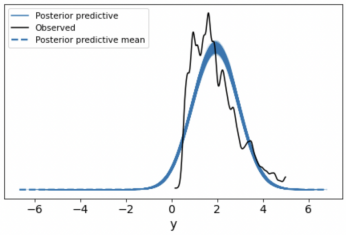


Figure 7: Posteriors over each parameter

	intercept	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	sigma
mean	1.9217	0.7045	0.1254	-0.1793	0.1494	0.0373	-0.1872	-0.8073	-0.7644	0.6032
sd	0.0044	0.0076	0.0049	0.0077	0.0056	0.0047	0.0045	0.0140	0.0136	0.0031
r_hat	1.0000	1.0000	1.0002	1.0001	1.0000	1.0000	1.0000	1.0000	1.0001	1.0000

(a) Summary of PyMC run



(b) PyMC Visualisation

Figure 8

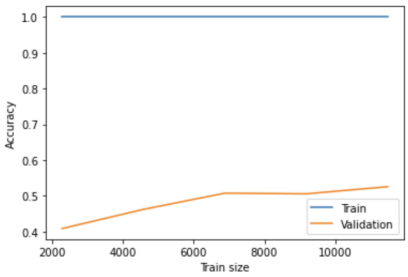
4 Trees and Ensembles

4.1 CART Decision Trees

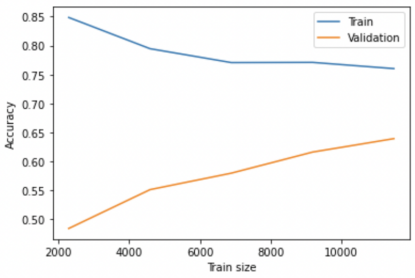
Tree-based models consist of a series of 'if-then rules' in a hierarchical structure which generate predictions for an input x by traversing the tree to reach a leaf node which assigns a label or value. Learning the tree structure involves determining which features to split on and then deciding when to stop splitting. Introduced by Breiman et al. [2], Classification and Regression Tree (CART) models are an approach where for each training example encountered, the best rule is found for splitting the observations by minimising an error function. Scikit-learn's decision tree regressor uses an optimised implementation of the CART algorithm (see algorithm 2).

Algorithm 2: Decision Tree Algorithm

1: Start from root node
2: Run exhaustive search over each possible variable and threshold for a new node. For each:
   a) Compute average value of target variable for each leaf of proposed new node
   b) Choose variable/threshold which minimise error if we stop adding nodes here
   c) Add new node for the chosen variable and threshold
3: Repeat 2 until there are n datapoints associated with each leaf
4: Prune tree to remove branches where error reduced $<$ tolerance value ϵ.

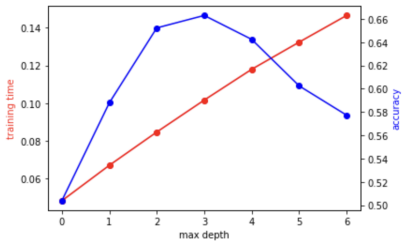


(a) Decision Tree training curves before tuning

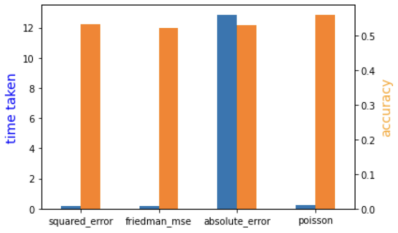


(b) Decision Tree training curves after tuning

Figure 9



(a) Effects of changing max depth



(b) Effects of changing criterion

Figure 10

Training a decision tree regressor with default parameters yields a training accuracy of 1.0 and a test accuracy of 0.61, with training curves shown in Figure 9a. While models are easy to understand and implement, decision trees tend to overfit on training data, resulting in misclassification of test examples. Tuning hyperparameters can alleviate this problem to some extent (Figure 9b). A grid search found that a maximum depth of 9 and a maximum features of 8 affords a training accuracy of 0.793 and a test accuracy of 0.698. It is intuitive that reducing maximum depth of a tree alleviates overfitting, since the tree cannot keep creating new branches to converge onto the optimal solution for the datapoints encountered. Due to the simple nature of trees, there are many possible structures which could give similar accuracies, resulting in high volatility of parameter tuning. For example, changing one small setting could see the grid search favour a far higher maximum depth value. This means that trees are an excellent tool for producing a 'good enough' model, but it is less intuitive to produce an 'optimal' model. Figure 10a shows the trade off between training time and accuracy in tuning maximum depth. Training times for the decision tree are so small that they can be considered negligable, however if the problem to solve was more complex it could be sensible to take into account the apparent linear relationship between this parameter and training time. Figure 10b compares criterions used for selecting which feature to split on. The absolute error criterion swamps the others in terms of time complexity, while only transpiring in an accuracy increase of 0.01 above the squared error and mse methods. The friedman_mse method is the optimal criterion to use in this case for the optimal efficiency-accuracy trade off.

A shortcoming of decision trees is their inability to predict outside the range of labels encountered, resulting in explicit upper and lower bounds on predictions. However, there are some sit-

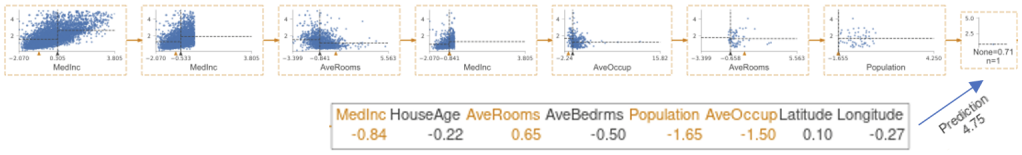
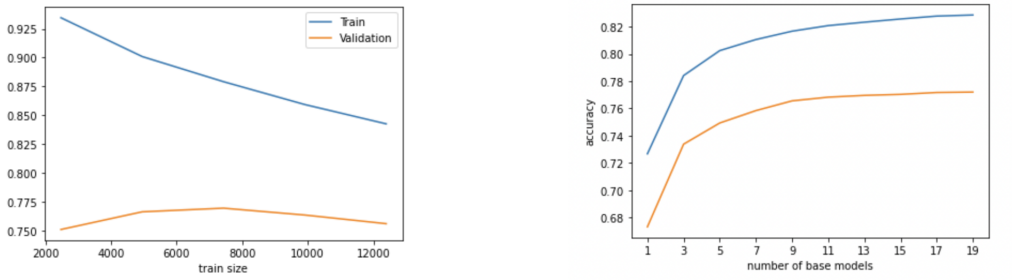


Figure 11: Path of point through tree



(a) AdaBoost training curves

(b) Base models effect on AdaBoost accuracy

Figure 12

uations where decision trees apply better than linear regression, for example if it is important to easily tell each parameter's influence on the prediction. In this case, it is simple to calculate that the 'MediCare' attribute sways the prediction most heavily. Iterating over test set predictions identifies the point for which the prediction differs most greatly from its true value. The highest difference was 3.74, which is substantial considering all points lie between 0.15 and 5.0. Figure 11 shows the traversal path of this point, which shows that the second node was pivotal in the misclassification since the MedInc feature of the example was close to the decision boundary of the node. This means that if this value were to differ even slightly, the point would have taken a totally different route down the tree from this point.

4.2 Ensemble Methods

An ensemble is a combination of models which can often perform much better than the average individual and sometimes even better than the best individual model. Boosting is a method which trains base models in sequence, ensuring each model addresses the weaknesses of the ensemble. Instead of training a new base model on a random sample, weights are set according to the performance of previous base models. AdaBoost is a popular implementation of this algorithm, which performs best on weak learners and has the key benefit of being able to boost almost any machine learning method. Figure 12a shows the learning curves of the AdaBoost algorithm applied to California Housing with the Decision Tree from section 4.1 as its base estimator. The result was a test accuracy of 0.758, an boost of more than 0.5 above the simple decision tree. Changing the number of base models affords an increase in accuracy as depicted in Figure 12b. The accuracies of ensembles methods evaluated were consistently higher than those gathered from Bayesian linear regression, which sat at a maximum of 0.610. Ensemble methods are more easily interpretable and thus can be tuned more easily and efficiently.

References

[1] Richard Ernest Bellman. *Dynamic programming*. Princeton University Press., 1957. ISBN: 978-0-691-07951-6.

[2] L Breiman. *Classification and Regression Trees*. Routledge, 1984. ISBN: 9781315139470.

[3] Daniel Golovin et al. 'Google Vizier: A Service for Black-Box Optimization'. In: New York, NY, USA: Association for Computing Machinery, 2017. ISBN: 9781450348874. DOI: 10.1145/3097983.3098043. URL: https://doi.org/10.1145/3097983.3098043.

[4] Matthew D. Hoffman and Andrew Gelman. *The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo*. 2011. DOI: 10.48550/ARXIV.1111.4246. URL: https://arxiv.org/abs/1111.4246.

[5] K. Lakshminarayan. 'Imputation of Missing Data Using Machine Learning Techniques'. In: *KDD* 96 (1996).

[6] Hanxiao Liu, Karen Simonyan, and Yiming Yang. *DARTS: Differentiable Architecture Search*. 2018. DOI: 10.48550/ARXIV.1806.09055. URL: https://arxiv.org/abs/1806.09055.

[7] Guilherme D. Pelegrina et al. 'Analysis of Trade-offs in Fair Principal Component Analysis Based on Multi-objective Optimization'. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. 2022, pp. 1–8. DOI: 10.1109/IJCNN55064.2022.9892809.

[8] Scikit-Learn. *Neural network models*. 2022. URL: https://scikit-learn.org/stable/modules/neural_networks_supervised.html (visited on 11/05/2020).

[9] SkLearn. *Fashion-MNIST*. 2022. URL: https://inria.github.io/scikit-learn-mooc/python_scripts/datasets_california_housing.html (visited on 11/05/2020).

[10] Rob Story. *Folium 0.12.1 documentation*. 2013. URL: https://python-visualization.github.io/folium/index.html (visited on 11/05/2020).

[11] Lipo Wang. *Support vector machines: theory and applications*. Vol. 177. Springer Science & Business Media, 2005.

[12] D.H. Wolpert and W.G. Macready. 'No free lunch theorems for optimization'. In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82. DOI: 10.1109/4235.585893.

[13] Zalando. *Fashion-MNIST*. 2017. URL: https://github.com/zalandoresearch/fashion-mnist (visited on 11/05/2020).