Machine Learning CW Report

Jakub Leszczynski
pt18419
University of Bristol

1.Introduction

This report focuses on the analysis of some of the methods that have been presented in the Machine Learning unit and explanation of the obtained results. The methods are used on the MNIST database of hand-written digits and the California housing regression dataset.

2. PCA

Principal Component Analysis, or PCA, is the process of computing the principal components of some dataset, such that the orthogonal projection of the data points onto the subspace of those principal components maximizes the variance of the projected points. In other words, PCA is a method that is often used to reduce the dimensionality of large data sets, increasing interpretability without losing too much information. We need variance as it describes the spread of our data. Therefore, the dimensions that have the highest variance, contain the most information about the differences in data points.

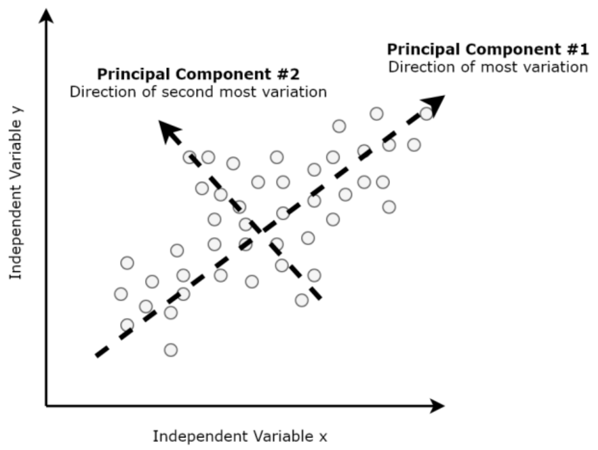


Figure 1. Illustration of variance captured by principal components [1].

Principal components are the eigenvectors of the covariance matrix ordered by eigenvalue [2]. They can be computed using the formula:

$$Su_1 = \lambda_1 u_1 \quad (2.1)$$

Where S is the sample covariance matrix, λ_1 is the eigenvalue and u_1 is the corresponding eigenvector.

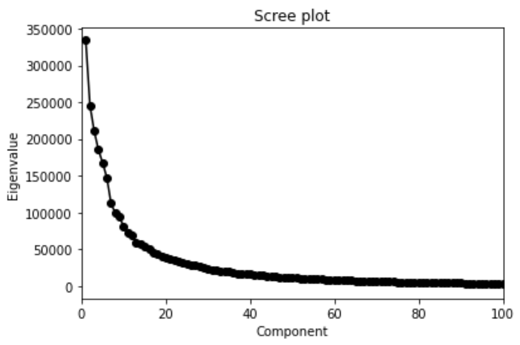


Figure 2. Scree plot of the MNIST dataset eigenvalues.

Dimensionality reduction is useful as it helps us better visualize the data and makes it easier to gain some understanding of it. Additionally, it makes further computations faster, and we need less space to store the data.

Thankfully, it is often the case that not all dimensions are important to understand the dataset. In fact, the first 87 principal components of MNIST are enough to explain 90% of the variance in the data.

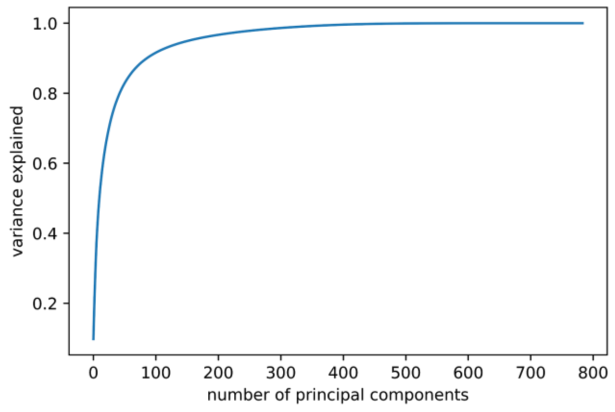


Figure 3. The relation between the amount of variance captured and number of components used in the PCA.

After running PCA on the MNIST dataset, we can plot the data using only the first two principal components.

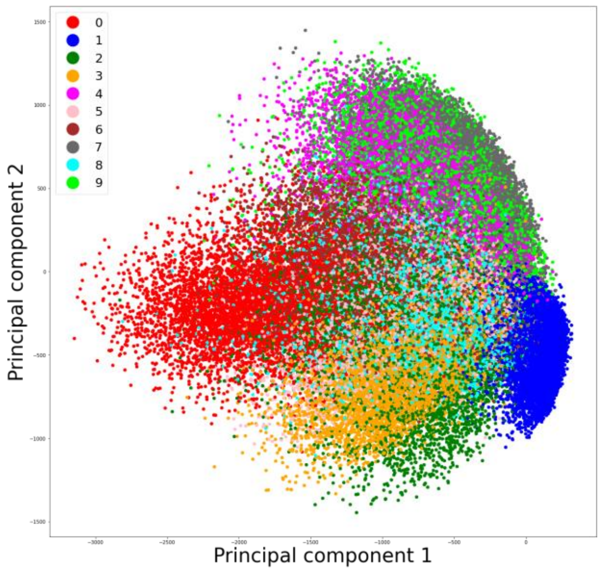


Figure 4. Plot using only first two principal components, each colour represents a different digit.

There is a lot of overlap between digit classes, only 0 and 1 are somewhat separated from the rest. The first two components can therefore give us some information about the data; however, it is not enough to give clear boundaries between classes.

Furthermore, to gain some understanding of what the principal components "do", we can visualise their weights.

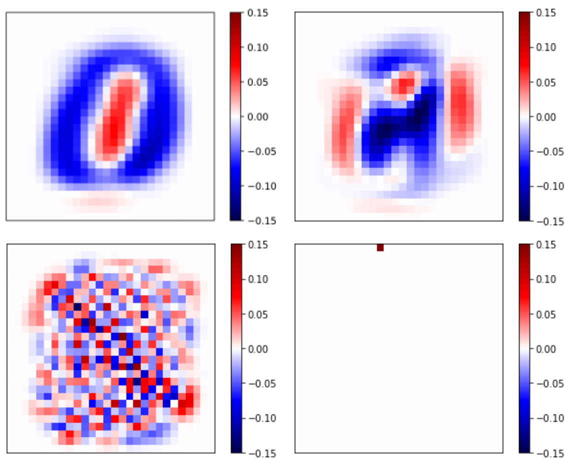


Figure 5. Visualisation of the weights of the 1st, 5th, 200th and 750th principal components, respectively. Generated using the code from [3].

The first principal component seems to be discriminating between 0 and 1, i.e., given a sample of 0 it will lie mostly on the blue circle and the dot product will give a small value for this component. On the other hand, a sample of 1, will lie mostly on the red line in the middle and the dot product will give a high value for this component. Therefore, we can expect that by using this component, there will be some separation between 0s and 1s, which can indeed be seen on the previous plot, where red 0s are to the left, relatively far from the blue 1s on the right.

However, with each subsequent principal component, the weights are harder to explain. The 5th component still seems to have some structure, but the 200th is already very chaotic and the 750th seems to care only about one pixel at the top of the images.

Lastly, we can reconstruct the data with a subset of principal components, without losing too much quality.

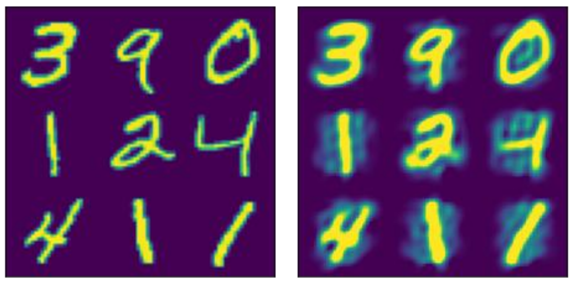


Figure 6. Composed 9 original images on the left, and their reconstruction using only first 50 principal components on the right.

3. K-Means

K-means Clustering is, just like PCA, an unsupervised machine learning method. It tries to learn an optimal division of the given data into a predefined number of clusters (K), such that data points in the same cluster are similar to each other. K-means Clustering consists of solving two optimisation problems [4]:

$$r_{nk} = \begin{cases} 1 \; if \; k = arg\,min_j \, \|\boldsymbol{x}_n - \mu_j\|^2 \\ 0 \qquad otherwise \end{cases} \tag{3.1}$$

$$\mu_k = \frac{\sum_n r_{nk}\boldsymbol{x}_n}{\sum_n r_{nk}} \tag{3.2}$$

$r_{nk} = 1$ if datapoint $\boldsymbol{x}_n$ is assigned to cluster k.
μ_k is the mean of cluster k.
In step (3.1) we assign each point to a cluster and in step (3.2) we recalculate cluster means. We repeat those steps until convergence.

We can run K-Means Clustering with random initial clusters on the MNIST data using only the first two principal components. The returned cluster labels are arbitrary, but we can calculate which class has highest occurrence in each cluster and treat it like a cluster of that class [5]. After that we can plot a confusion matrix and calculate the accuracy of clustering. In this case the accuracy score is quite bad (around 40%), which should not be a surprise as we used only a very small subset of dimensions. Nonetheless, on the confusion matrix we can see that, compared to other classes, 0s and 1s are relatively well labelled. In a perfect clustering all values except of the first diagonal would be 0.

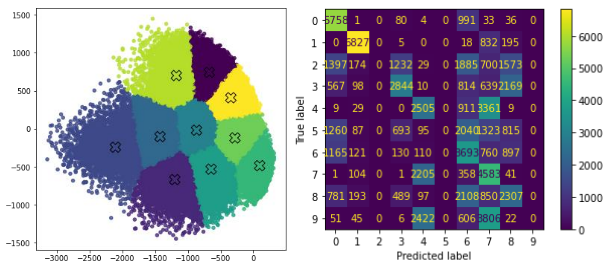


Figure 7. K-Means Clustering on first two principal components of the MNIST dataset and the confusion matrix of the results.

Running K-Means on all the dimensions gives better results, with accuracy around 58%. However, this is still not a great result.

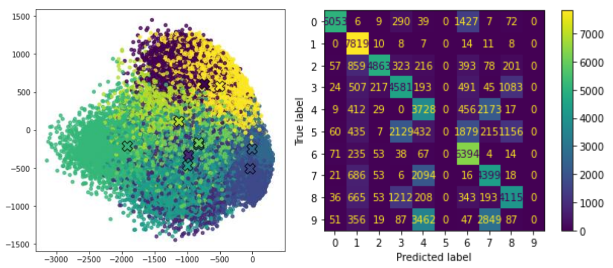


Figure 8. K-Means Clustering on all principal components of the MNIST dataset plotted on the first two components and the confusion matrix of the results.

4. Artificial Neural Networks

Artificial Neural Networks are a collection of interconnected nodes or units. They are loosely based on the animal neuron. The processing power of ANNs comes from the weights of connections between units, which are learned from the training data [6].

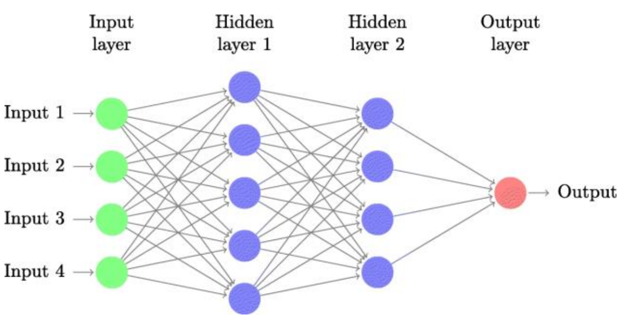


Figure 9. A schematic ANN, with 2 hidden layers and one output node, figure from [7].

This method is an example of supervised learning. In the training process, we provide the labelled training data, the model learns the patterns in the data and infers a function which can then be used to match labels in new unlabelled data.

For this task we are again using the MNIST dataset so in our case the input layer has 784 nodes, one for each dimension and 10 output nodes, one for each possible class. First, I ran the MLPClassifier with one hidden layer with 64 neurons with relu activation function and stochastic gradient descent solver. Then plotted the loss value against iterations. The smaller the loss, the better a job the classifier is doing at modelling the relationship between the input data and the output targets.

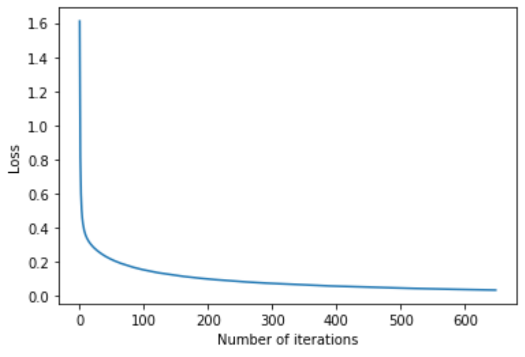


Figure 10. Loss function of the ANN.

This shows us that the model is learning the patterns in the training data quickly. However, this does not tell us how well the model predicts labels in new data. Therefore, I plotted a confusion matrix on the test set.

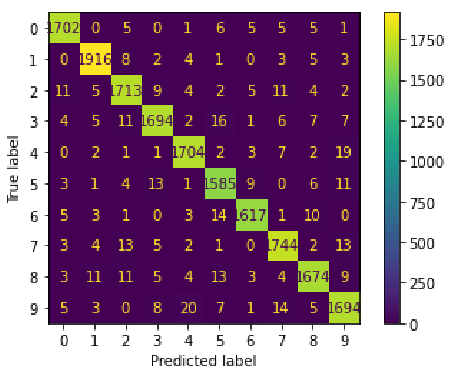


Figure 11. Confusion matrix on the test set.

The accuracy is high, (0.971) but lower than in the training data (0.993). We can further visualise the difference in accuracy by plotting the accuracy score over epochs.

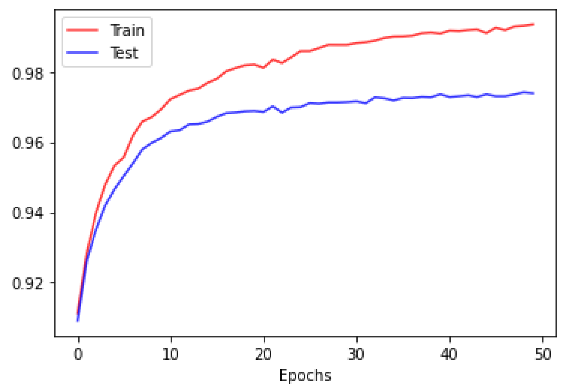


Figure 12. The accuracy of our model over time, figure plotted with adapted code from.

This shows us that our neural model is moderately overfitting. It follows the patterns in the training data too closely, which produces errors in new unseen data.

The results so far have been influenced by the choice of the hyperparameters of the model. It is worth considering how the accuracy changes when we use different configurations of the model. The search space of all the possible combinations of the hyperparameters is huge, so let us focus on different numbers of neurons and learning rates.

By investigating the effect of the constant learning rate, we can see that if we choose a bigger value, then the model will have some trouble converging. The model becomes unstable, during gradient descent, it fails to find the exact minima, as the jumps are too big.

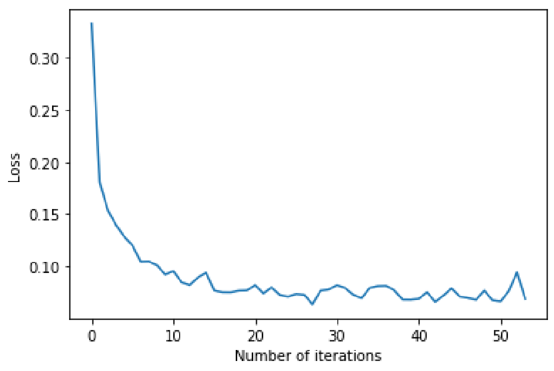


Figure 13. Loss function of our model with a higher value of learning rate.

One way to deal with this problem is to use an adaptive learning rate. In this case, after some iterations where the model does not improve, the learning rate is lowered, and the gradient descent manages to find its way without overshooting.

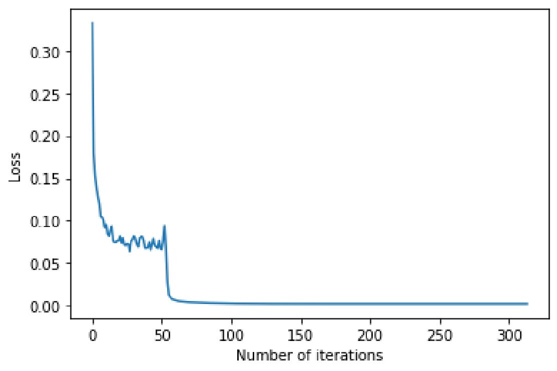


Figure 14. Loss function of our model with adaptive learning rate.

To find a better configuration of hyperparameters I used the GridSearchCV library. The results show that by increasing the number of nodes to 150 the accuracy goes to 0.97697, which is already slightly better than original. Next, testing some learning rates showed that testing accuracy increased further to 0.97949.

Further optimisations, for example choosing a better alpha (regularisation) value or increasing the number of layers, are possible. Unfortunately, finding more optimal values is tremendously expensive computationally.

Lastly, we can try to understand what the ANNs is looking for by visualising the weights of some example neurons.

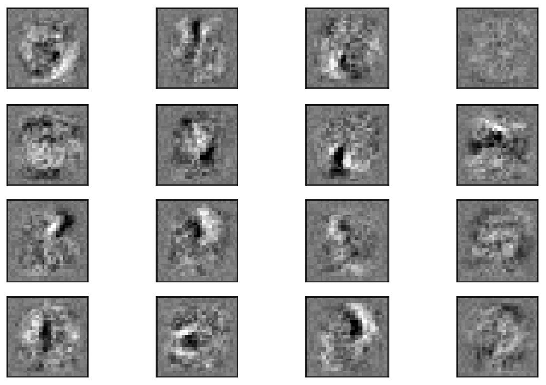


Figure 15. Visualisation of the weights of 16 neurons from our model.

Unfortunately, this is not very informative. The "black box" nature is one of the disadvantages of Artificial Neural Networks, i.e., it is very hard to understand what makes ANN come to the prediction that it makes [8]. Another disadvantage is that ANNs are computationally expensive, it takes them a long time to learn the patterns in the data [8].

Additionally, for the image classification problems, Convolutional neural network architectures show superior performance as they consider neighbourhoods of pixels and not only their values.

5. Support Vector Machines

SVMs like ANNs are supervised learning models, but opposite to them SVMs are non-parametric, i.e., the number of support vectors depends on the data. SVMs are fundamentally two class classifiers, the algorithm plots each data point in n-dimensional space, where n is the number of dimensions and finds a hyper-plane that differentiates between two classes of those points. In this part we will again train and test our models on the MNIST dataset. However, this dataset has 10 different classes. Thankfully, scikit-learn automatically uses one-versus-one or one-versus-the-rest approaches for us, depending on the kernel used. Nevertheless, this means that under the hood, we have to fit 10 different SVMs in order to differentiate between classes, whereas we only needed a single neural network. This means that the more classes there are, the more computationally costly it is to use SVMs.

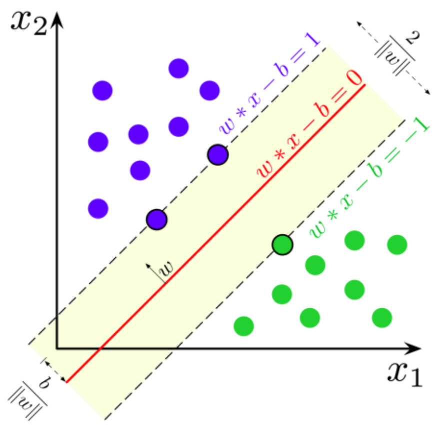
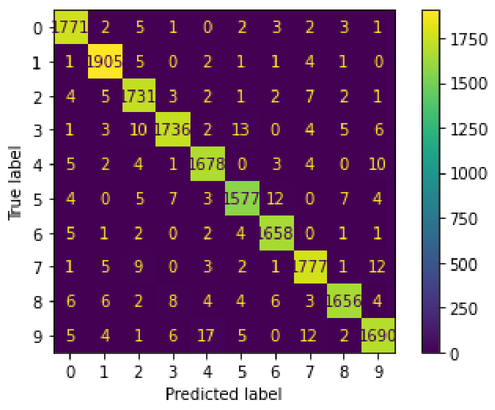


Figure 16. Hyperplane dividing two classes. Samples on the margin are called the support vectors [9].

Training an SVM with linear kernel results in 0.973 train and 0.934 test accuracy scores. However, only changing the kernel to RBF (Radial Basis Function) already improves the accuracies to 0.989 and 0.978, on train and test sets, respectively. We will therefore focus on the second.

After some tuning of hyperparameters for the rbf kernel like gamma, which defines the influence of training examples, or C value, which controls the cost of misclassification, our accuracy can increase to 0.981. Although this looks even better than our ANN model, the problem is that we did not check all possible combinations of hyperparameters for ANN.



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	1771	2	5	1	0	2	3	2	3	1
1	1	1905	5	0	2	1	1	4	1	0
2	4	5	1731	3	2	1	2	7	2	1
3	1	3	10	1736	2	13	0	4	5	6
4	5	2	4	1	1678	0	3	4	0	10
5	4	0	5	7	3	1577	12	0	7	4
6	5	1	2	0	2	4	1658	0	1	1
7	1	5	9	0	3	2	1	1777	1	12
8	6	6	2	8	4	4	6	3	1656	4
9	5	4	1	6	17	5	0	12	2	1690

Figure 17. Confusion matrix using rbf kernel, with gamma=0.01 and C=10.

A problem with SVMs, is that for each prediction of a new datapoint, the kernel-function has to be called with each support vector, which leads to magnitudes lower prediction speed than neural networks [10]. The time it takes to predict labels in our test set with rbf kernel SVM is around 3min15sec and with our ANN model it was only around 0.13sec.

On the other hand, it can be proven that SVMs, given enough time and precision, do not get stuck on local minima [11], which is a problem for ANNs.

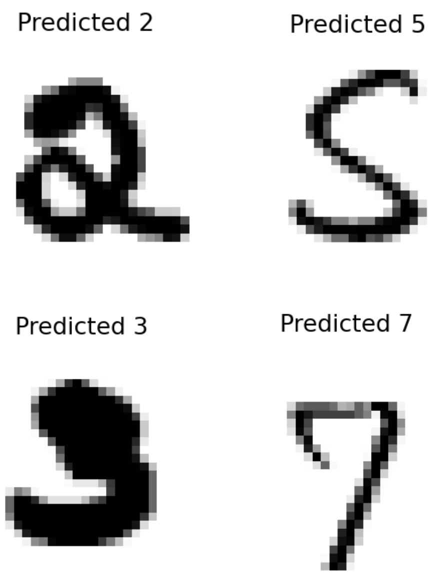


Figure 18. Some examples of the SVM model predictions on the test set.

6. Bayesian Linear Regression

From this part onward, we will focus on the California housing dataset with the goal being to predict the median house values and because they are continuous values, this is a regression problem.

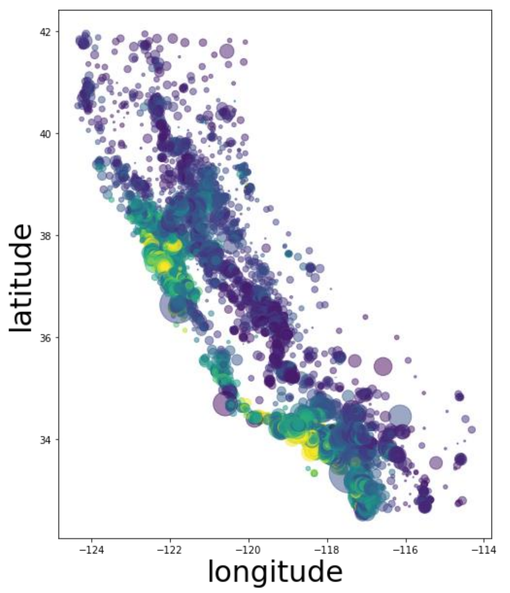


Figure 19. Scatter plot of data using longitude and latitude coordinates. The scale on the right describes the median house value in 100,000$.

As this is an example of real data, we can expect some data cleaning to be necessary before analysis.

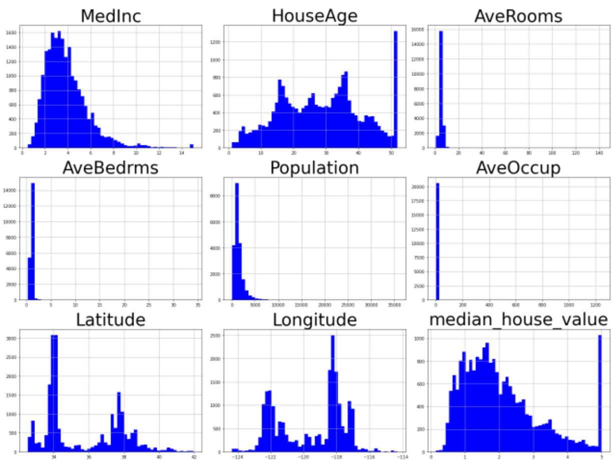


Figure 20. Histograms of all variables from the California housing dataset.

We can already see that both the median house values and the house age histograms have suspicious looking spikes at their highest values.

Value	Count
5.00001	965
5.00000	27
4.75000	8
4.83300	6
4.66700	4

Value	Count
52	1273
48	177
50	136
49	134
51	48

In the left table we can see that the count of median house values that were equal exactly 500001$ (the values are divided by 100,000) is much higher than the previous values. We can observe the same thing in the House Age count. Those values might be outliers or wrong data, so I decided to clip them from the dataset.

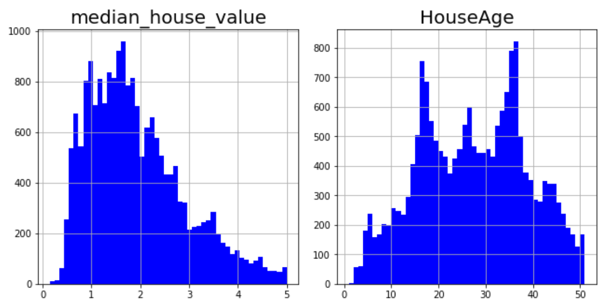


Figure 21. Updated variable histograms.

The next observation is that Average Bedrooms, Average Rooms and Average Occupancy features all look like simple spikes that do not carry much variance and are not valuable because of that. Therefore, I introduced two new variables instead of those three. First one being Average Bedrooms divided by Average Rooms with the assumption that in more expensive houses there are more bedrooms and the second being Average Rooms divided by Average Occupancy with the assumption that more expensive houses have more rooms in general per person living there.

Another insight we can have is to see the correlations between different features and the target. This shows us that the median income in the block should have a strong influence on the house values in that block.

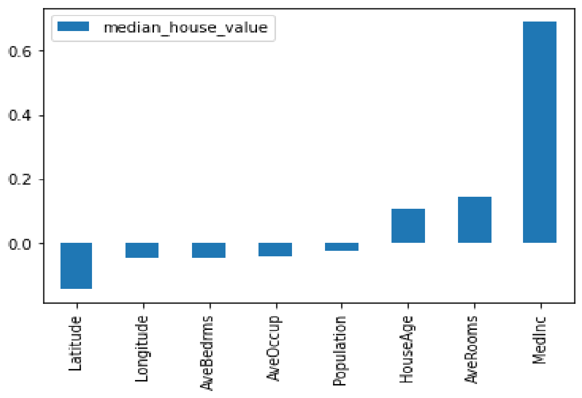


Figure 22. Correlation between different features and median house value.

Now having the prepared data, I want to first fit a linear model to the data using only the median income variable. In a standard least squares regression model, the formula is:

$$y = \alpha + \beta x + \epsilon$$

Where alpha is the intercept, beta is the coefficient, and the epsilon is the noise in our data. After fitting the data, this model has fixed values of those parameters. However, in the Bayesian approach we assume that our data comes from a probability distribution described by those parameters, i.e.:

$$y \sim Normal(\alpha + \beta x, \sigma^2)$$

With $\alpha + \beta x$ being the mean of the distribution and σ being the standard deviation. Moreover, all of those parameters are assumed to come from their own distributions as well. The goal of Bayesian approach is therefore to find the posterior distributions of the parameters, the formula for the Bayesian inference we use is:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}$$

The posterior distribution of the data $p(\theta|D)$ is dependant on the likelihood given data $p(D|\theta)$ and prior distributions $p(\theta)$ normalised by the "evidence" $p(D)$ [12].

The approach to obtain the posterior is to use a sampling algorithm that with every sample will gradually improve the accuracy of the distribution. For this task I will be using the No U-Turn Sampler.

Last thing to do is split the data into training and testing samples and standardize the features by removing the mean and scaling them to unit variance using StandardScaler sklearn library. This will greatly reduce the time needed for sampling

So, in my initial model I set the intercept and the coefficient priors with Normal distribution, and for noise I used the HalfNormal, all with 0 mean and 1 standard deviation. Then I run NUTS sampler with 1000 steps, using the train median values as target. After that I obtain 1000 samples for each point in the train data from the posterior distribution. Taking the mean of them gives me my linear regression line.

To test the accuracy of the models I will be using mean squared error. The lower the error the better fit we have.

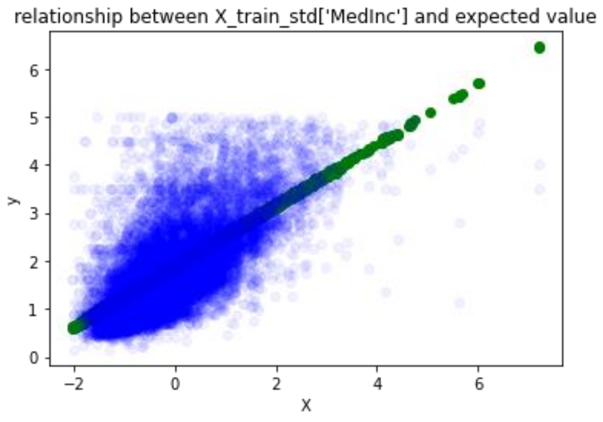


Figure 23. Scatter plot of median income and median house values and a green linear regression line.

The error on the train data is around 71,850 and by using Theano library's shared function, I can calculate that the error on the test data is very similar, so our model is not overfitting. We can also plot the distributions of the parameters of the model to see what means and standard deviations they have.

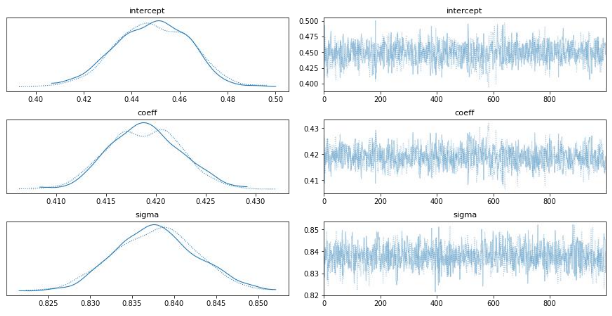


Figure 24. The distributions of the parameters of the model.

After adapting the model to consider all variables, with each coefficient coming from a normal distribution, and fitting the data the calculated error is around 60,000. However, I used Bayesian inference to generate same regression line that I could get by standard linear regression. Actually, fitting the data on LinearRegression module from sklearn gives same results, with almost identical error, so what is the advantage of using Bayesian approach?

The benefits of our model are coming from the posterior distribution. In standard linear regression, the calculated parameters are fixed, but in Bayesian case they have their own distributions, which can tell us how confident we are about our model. For example, additionally to the regression line we can plot the error bar at each point, which tells us what the range of influence of the noise in our data is.

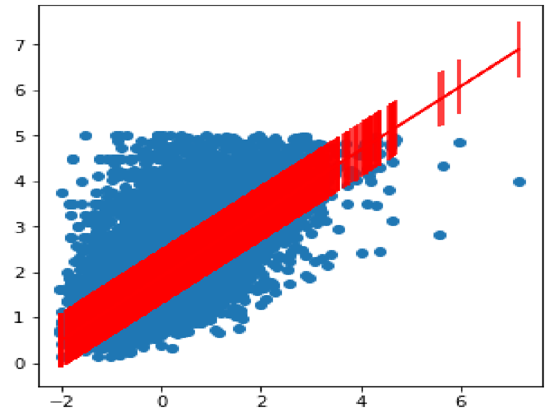


Figure 25. linear regression with an error bar.

Moreover, by sampling from the posteriori, we can visualise the influence of each variable on the final median value.

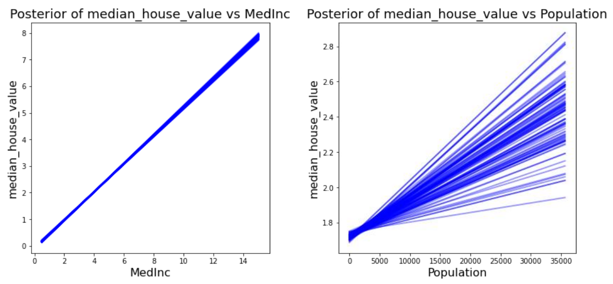


Figure 26. Plots of 100 different samples from the posterior distributions. Each line represents a different sample [13].

We can see that median income has a strong linear relationship with the target i.e., each line is similar. Conversely, we cannot be sure what is the influence of the population feature. When the population is small, all samples are similar, but the higher the value the bigger the variance. Although there is some linearity in all the samples, we are unsure about the exact slope, there is some noise in this particular feature. However, this might be because there were not enough samples, and the distribution did not converge yet. When more data comes the model will update the probabilities, which is the strength of Bayesian models.

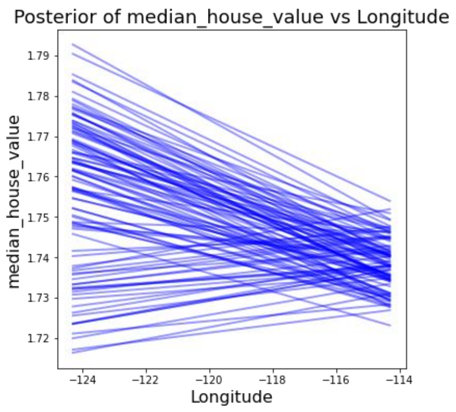


Figure 27. Samples from the Longitude posterior [13].

On the other hand, longitude is a feature that we cannot even say if it has a positive or negative correlation with the target. Some samples tell us that the median house value should go down when the longitude goes up, but some say exactly the opposite.

So far, I have been using normal distributions for all the parameters, but this might not be always be the true distribution. We can think what other distributions would better fit our data.

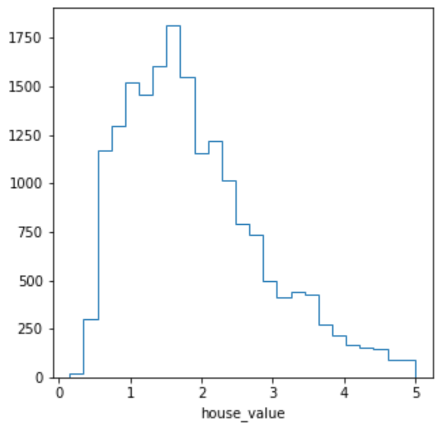


Figure 28. Histogram of median house values.

In fact, the distribution of our target looks more like a Gamma distribution. Running an Anderson-Darling test can tell us that with p=0.01 we can say that it is not a normal distribution. Therefore, in my final model I used Gamma distribution for the noise parameter, but similar analysis could be done for all other priors.

6. Random Forests

Random Forest is an ensemble of decision trees. It is created in a process called bagging. Each decision tree in the forest is trained on a different sample of the training set. For prediction, the model takes the mean (for regression) or majority vote (for classification) of all individual predictions of the component decision trees. It is a supervised learning algorithm. In this section I will fit and analyse a sklearn RandomForestRegressor model on the previously prepared California housing dataset. Immediately after fitting the model, without any hyperparameter tuning, the mean squared error on the test set is only around 45000, that is better than the Bayesian linear model by 15000, which is a huge improvement.

Random forests have many hyperparameters that we could optimise. First, I visualised the importance of depth that the decision trees can reach.

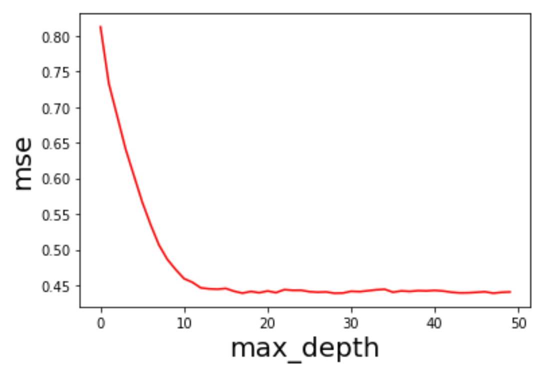


Figure 29. Relation between max_depth and error.

From the plot we can conclude that shallow trees perform worse, but after the depth of around 10, there is not much improvement. GridSearchCV using 10 cross validations on a range from 1 to 50, tells that the best parameter of depth is 48.

This improves the accuracy of the model, but it does not consider the time needed for training. We can plot this trade off. Again, after reaching some threshold, the models oscillate around same value.

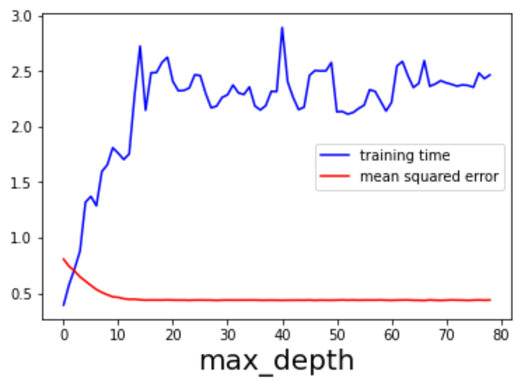


Figure 30. Training time with different max_depth and error.

Another hyperparameter that we can inspect is the number of estimators. Estimators are the individual decision trees that the forest consists of.

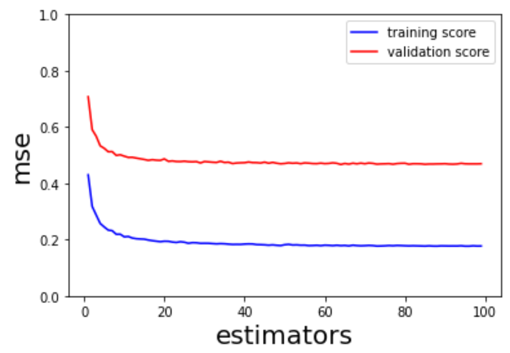


Figure 31. Number of estimators and the error.

Increasing the number of estimators does slightly lower the error, but it also impacts the computational complexity.

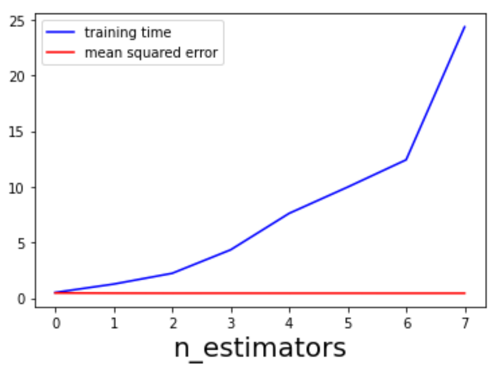


Figure 32. Training time with different number of estimator and error, the x axis is in hundreds.

Therefore, the choice in this trade off depends on how much we value accuracy and how much computational power we can spare for the task. For my model I used the value of 300.

Last hyperparameter I will take under consideration is the number of features that the decision trees consider when looking for a best split.

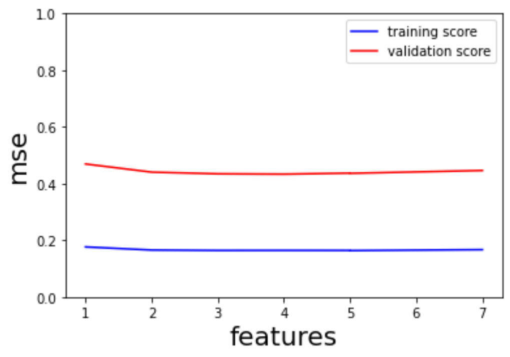


Figure 33. Number of considered features and the error

We can see that there is not much variance in the error rates, and although the GridSearchCV using 10 cross validations returns the best number of features as 4, it is not significantly important.

If we change the max depth to a small value, we can visualise a single decision tree. We can gain some insights in the decision process by computing the impurity that the features remove in nodes, but this does not tell us much about individual predictions on test data. However, for all decisions being made, there is a path from root to the final leaf that contains valuable information. A useful tool that can be used to extract this information is the Treeinterpreter library. It allows to track the prediction and record the value changes along the path and the features that cause them [14]. The final decision can be interpreted as a sequence of feature contributions and the overall bias of the model. This can be extended to a whole forest prediction by taking the mean of all individual predictions

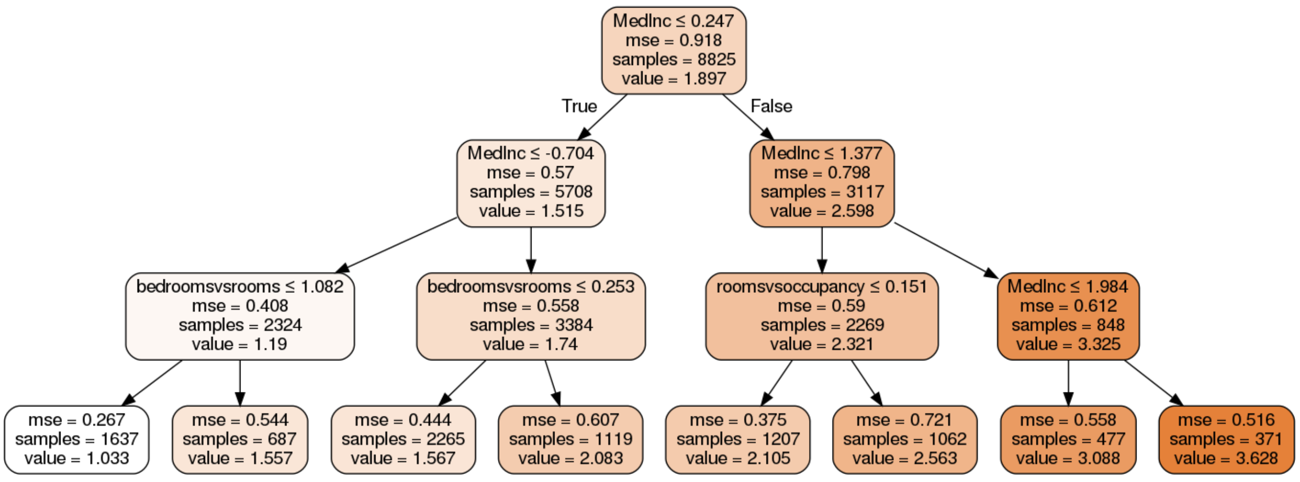


Figure 34. A single estimator, i.e., decision tree of our model, with depth 3.

7. Stacking

Now that we have a range of estimators, we can further improve predictions using stacking. Stacking consists of base models, which are trained on the data, and another meta-regressor that is trained on the outputs of the base models and calculates the final prediction. I created a wrapper class for my pymc3 model, so that I can use it for stacking. The smallest mse with pymc3 was 60000 and with random forest with best configuration it was around 44000. Using only those two models as base estimators and the default RidgeCV final estimator immediately slightly improves the result to approximately 43500. This shows that stacking is a relatively easy method that can boost the accuracy. Moreover, adding more random forest, with different configurations, as base estimators can improve the results even further. For example, base estimators consisting of 4 random forests with different max depths and numbers of estimators, and the addition of pymc3 model reduces the mse to 42500, which is the best result achieved by any of the models that I used.

However, it is not the case that stacking always brings improvement. First of all, the results greatly depend on the choice of final estimator, e.g., using a simple decision tree or another random forest for that purpose can actually give worse results than the best base model on its own. Additionally, stacking performs better when the base models' predictions are not correlated, i.e., the base models are different algorithms so that the errors in predictions that they produced have different underlying cause. Stacking can use this diversity to learn another algorithm that can "smooth" these errors and produce a better prediction. Simply calculating the mean of our base predictions would at best be equal to the best prediction, so there would be no improvement at all.

To better understand what final estimator is using for training we can use a decision tree with a small depth as the final estimator and visualise it.

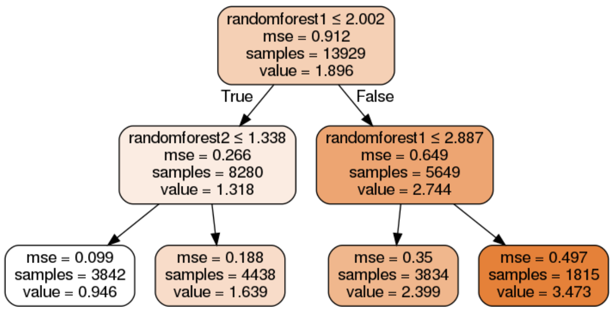


Figure 35. Decision tree with max_depth=2 as final estimator.

We can see that the variables used by this decision tree are actually the outputs from the base models, here randomforest1 and randomforest2.

8. Conclusion

In this report I investigated machine learning methods presented in the unit and analysed the results obtained by using them on the MNIST and California Housing datasets. The explanations, along with included figures are meant to demonstrate the understanding of the material. The coursework proved to be a valuable experience that broadened my knowledge about machine learning algorithms and will help me in my future data analyses.

References:

[1] ‘Principal Component Analysis—Unsupervised Learning Model | Hacker Noon’. Available: https://hackernoon.com/principal-component-analysis-unsupervised-learning-model-8f18c7683262 (accessed Dec. 10, 2020).

[2] C. M. Bishop, *Pattern recognition and machine learning*. New York: Springer, 2006.

[3] ‘Google Colaboratory’. https://colab.research.google.com/github/NeuromatchAcademy/course-content/blob/NMA2020/tutorials/W1D5_DimensionalityReduction/student/W1D5_Tutorial3.ipynb (accessed Dec. 10, 2020).

[4] J. Cussens, ‘COMS30035, Machine learning: k-means and mixtures of Gaussians’, p. 10, [Online]. Available: https://uob-coms30035.github.io/JamesLectures/gaussmix.pdf.

[5] S. Salaria, ‘K Means Clustering for Imagery Analysis’, *Medium*, Aug. 23, 2019. https://medium.com/datadriveninvestor/k-means-clustering-for-imagery-analysis-56c9976f16b6 (accessed Dec. 10, 2020).

[6] K. Gurney, ‘Introduction to Neural Networks’. Taylor & Francis, Oxford.

[7] ‘Artificial neural networks improve and simplify intensive care mortality prognostication: a national cohort study of 217,289 first-time intensive care unit admissions | Journal of Intensive Care | Full Text’. https://jintensivecare.biomedcentral.com/articles/10.1186/s40560-019-0393-1/figures/1 (accessed Dec. 10, 2020).

[8] ‘4 Reasons Why Deep Learning and Neural Networks Aren’t Always the Right Choice’, *Built In*. https://builtin.com/data-science/disadvantages-neural-networks (accessed Dec. 10, 2020).

[9] Larhmam, *English: Maximum-margin hyperplane and margin for an SVM trained on two classes. Samples on margins are called support vectors.* 2018. Available: https://commons.wikimedia.org/wiki/File:SVM_margin.png (accessed Dec. 10, 2020).

[10] ‘Support Vector Machines vs. Neural Networks - Indiji’. https://web.archive.org/web/20120304030602/http://indiji.com/svm-vs-nn.html (accessed Dec. 10, 2020).

[11] R. Berwick, ‘An Idiot’s guide to Support vector machines (SVMs)’, p. 28. Available: http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf (accessed Dec. 10, 2020).

[12] R. P. Costa and D. Damen, ‘COMS21202: Symbols, Patterns and Signals Probabilistic Data Models’, p. 30. Available: https://uob-coms21202.github.io/COMS21202.github.io/RuiLectures/Lec4-handout.pdf (accessed Dec. 10, 2020).

[13] W. Koehrsen, ‘Bayesian Linear Regression in Python: Using Machine Learning to Predict Student Grades Part 2’, *Medium*, Apr. 21, 2018. https://towardsdatascience.com/bayesian-linear-regression-in-python-using-machine-learning-to-predict-student-grades-part-2-b72059a8ac7e (accessed Dec. 10, 2020).

[14] ‘Random forest interpretation with scikit-learn | Diving into data’. https://blog.datadive.net/random-forest-interpretation-with-scikit-learn/ (accessed Dec. 10, 2020).