Machine Learning Implementation Report

Forename Surname – Raheel Shaikh

Word Count: 3,420

1. Executive Summary

Using the customer data of a company, we will be implementing various machine learning algorithms in the following tasks to predict the salary based on various attributes of the data. For each task, we will be predicting the salary of the customers based on the other attributes. We will be experimenting and comparing the results of various models to get the one with the best performance without overfitting the data. We will be using the mean squared error to calculate the loss function and evaluate our models. In task 6, we will be clustering the data into groups and visualising our results to draw insights from the plot.

2. Introduction to Machine Learning

The goal of machine learning, a subfield of computer science, is to make it possible for computers to "learn" without being explicitly taught. Its roots are in the 1950s artificial intelligence movement, and it places a strong emphasis on useful goals and applications, including prediction and optimization. In machine learning, computers "learn" through becoming more efficient at tasks through "experience."

Since "experience" in real-world applications typically refers to fitting to data, there is no apparent distinction between machine learning and statistical methods. In fact, whether a methodology is deemed "machine learning" or "statistical" depends as much on its history as on real differences, and depending on who you ask, many algorithms (such as least absolute shrinkage and selection operator (LASSO), stepwise regression, etc.) may or may not be deemed machine learning. In general, machine learning prioritises prediction accuracy above hypothesis-driven inference and typically concentrates on big, high-dimensional (i.e., containing many variables) data sets (Ki,2006).

Machine Learning can be broadly classified under “supervised” or “unsupervised”. In supervised machine learning, the outcome's value, also known as the dependent variable's "label" is known for each observation. Labelled data are those that have result values that are known. Standard statistical methods like linear and logistic regression as well as many of the most widely used machine learning algorithms are examples of common supervised learning techniques (e.g., decision trees, support vector machines). In unsupervised learning, the algorithm looks for logical connections and groupings within the data without considering any results or the "correct answer" (Duda et al., 2000). Statistical methods that aim to find undefined subgroups with comparable properties and unsupervised learning methods have similar objectives and structures (Bartholomew et al.,2011). Implementations of unsupervised learning that use clustering algorithms to group observations based on shared data properties are quite prevalent. Examples include expectation-maximization clustering with Gaussian mixture models and k-means clustering (Hennig, 2015).

3. Regression

Importing all the necessary libraries and packages like pandas, numpy, sklearn.preprocessing, sklearn.model_slection, and sklearn.linear_model to implement linear regression.

Data Pre-processing.

Data pre-processing refers to the procedures we must follow to alter or encode data so that a computer can quickly and readily decode it. The algorithm's ability to quickly analyse the properties of the data is essential for a model to be accurate and exact in its predictions. Here is how we perform data pre-processing.

First, we read the data "Comp1801CourseworkData.csv" using the pd.read_csv function provided by the Pandas library and stored the data into a variable called "raw_df." By doing this, we convert the data from a .csv (comma-separated value) format to a data frame with rows and columns, which makes it much easier for analysing and working with the data.

Next, we shuffle the data and assign it to a variable "df." This is to ensure that changes made to a model by each data point are independent and not biased by the same point that came before them. In the data set provided to us, we have columns with categorical values (example: education, work type, gender, and region) and the data is in the form of text; such values cannot be used while performing regression, so we make use of an encoding system that replaces the text with numbers. This is done by grouping the similar texts and assigning a unique value to them, and then replacing that text value with the corresponding digit in the entire column in the data set. By doing this, we convert the data into a machine-readable form. We will be using the label encoder to perform the encoding as it doesn't affect the dimensionality of the data, unlike one-hot encoding.

Next, we set our target variable "tar" to "salary," as this is the attribute for which we will be carrying out our prediction. We create another variable, "col" and assign a list of the remaining attributes. By doing so, we can easily include features that are relevant and remove those that do not contribute to the model's performance, but in our case, we will be using all the features. We then initialise two Numpy n-dimension arrays and call them "X_raw" and "y" where "X_raw" is our feature matrix used for predictions and "y" is the target array we want to predict.

Data Splitting

Data splitting is frequently used in machine learning to prevent overfitting. It is a case when a machine learning model fits its training data too well and is unable to consistently fit new data. We split the data into a 60:20:20 split, i.e., training data, which is used to train our model. Validation data, using which we fine-tune our model, and testing or unseen data that we use to evaluate our model.

We can go ahead and fit the model using the data at this stage, but this results in a poorly fitted model with an unsatisfactory evaluation matrix. We perform additional pre-processing operations to improve the model.

Polynomial features

We can discover possible new relationships between the features and the target and enhance the model's performance by creating polynomial features. To do so, we make use of the function "polynomialfeatures()" from the pre-processing libraries we included earlier. This results in raising an existing attribute to a degree (exponent); for example, if we have a feature "x" the polynomial feature will be "x2" with degree = 2.

Feature scaling

To further improve the performance of the model, we can perform feature scaling. What this does is standardise the independent attributes in the data set to vary within a fixed range. This can be done by using the "StandardScaler()" function.

Initially, we do not know which is the best degree for our model, so trying our model for various degrees would be a repetitive task for us. We can implement a "for" loop in our code and set the highest degree that we want to test for. This experimentation is carried out using linear regression. We also perform feature scaling, i.e., fitting and transforming our training data to be standardised. We plot a graph as shown in figure 1 and get a degree of 2 for which the MSE is the least. As we can also see in the plot, the lines start to deviate from each other at degree 3 and keep increasing; this indicates overfitting of the data for higher degrees of polynomials.

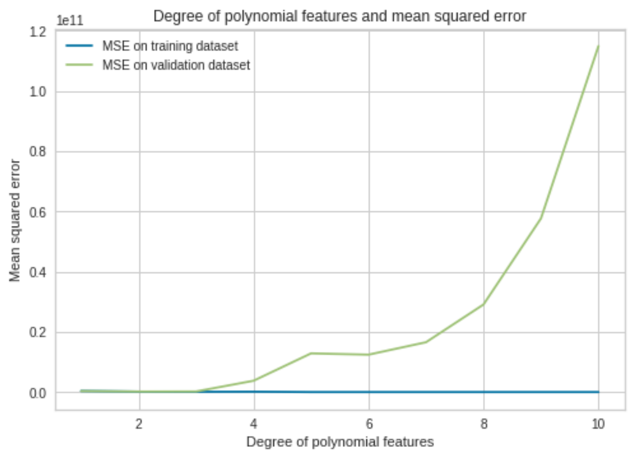

Figure 1: MSE vs polynomial degree

Fitting our model using Random Forest algorithm

We now fit our Random Forest model by using the RandomForestRegressor() function provided by the sklearn.linear_model package, and we set the "n_estimators" value to 200. We pass our training data as parameters, i.e., "X_train" and "Y_train."

Prediction and Evaluation Matrix:

Now we perform prediction using the "model.predict()" function also provided by the "sklearn.linear_model" package. We do this by passing first the "X_valid" (validation data) as a parameter and then computing the evaluation matrix for our output, i.e., the Mean Squared Error (MSE). We do the same with the test data and compare both the MSEs; in our case, the MSE for the test data is less than the MSE for the validation data, so we can safely conclude that our model is not overfitting. We get an R2 score of 0.8509, which tells us that our model is able to explain 85.09% of the variability of our target variable. In other words, our model has 85% accuracy.

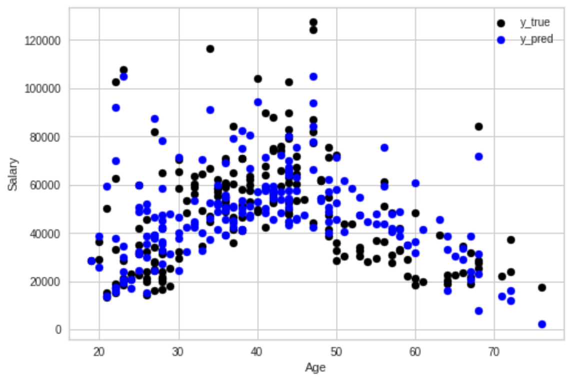

Figure 2 :Salary prediction using age

From figure 2 we can say that are model had a decent fit as most of the blue dot lie closely near the black dot thus, showing a good accuracy in the prediction.

After experimenting with different regression model like Linear Regression, Ridge Regression and Lasso Regression we the following results.

	MSE of validation data	MSE of test data	R2 score
Linear Regression	171248860.674	142684141.398	0.7272
Ridge Regression	171175765.651	142725476.282	0.7271
Lasso Regression	171247279.317	142684570.627	0.7272

Random Forest	92239509.417	78032950.491	0.8508

Table 1: MSE evaluation of the linear regression models

We perform the evaluation of all the models based on the MSE. The MSE is nothing but the sum of the squared differences between the actual values and the values that our model predicted. As we can see, the evaluation results for Ridge and Lasso are similar to those of Linear Regression, as they belong to the same family of regressors. The Random Forest has the highest R2 score and the lowest MSE, and thus proves to be the best regression model.

4. Binary Classification

Binary classification is a supervised learning approach used in machine learning that divides incoming observations into one of two classes. The 0 and 1 columns represent two potential classifications for each observation in the following examples of binary categorization. In our case, we have to classify the salary as either greater or less than 35000.

Data pre-processing

Since our data has some columns with a categorical value, we make use of an encoding scheme so that our data is in a form suitable for our model fitting. We first perform label encoding on the "Sex" column using LabelEncoder(). Next, we implement One-Hot encoding on the "region," "education," and "work type" using the OneHotEncoder(). In the LabelEncoding scheme, categories are assigned a number, and the value of a specific category for a particular column is replaced by that number. Whereas in label encoding, we create new columns for the different categories and add 1 if that datapoint belongs to that category or 0 if it does not. Here, after adding the new columns, we drop the column with the categorical values.

Next, we select the features for training and testing our data; in our case, we are using all the features. The salary column is then modified by inserting 1s if the salary is greater than 35000 and 0s if the salary is less than 35000.

Splitting of Data

We need to split the data in order to carry out training, validation, and testing. We use the 60:20:20 split, and by doing so, we can make sure we are not overfitting our model. We then repeat the polynomial featuring with degree = 2 and standardisation steps to ensure that the model is not biased toward a specific value.

Fitting the model (using SVM)

Using the Support Vector Machine (SVM) algorithm to fit our model available in the "sklearn.svm" package Now we need to set the hyperparameter so that we get the best accuracy, so we run a grid search that runs the SVM algorithm for all the combinations of the hyperparameters and get C = 1000, which is the penalty for the misclassified points. The higher the "C" value, the more the SVM attempts to minimise misclassifications due to a high penalty. The gamma value is 0.001, and the kernel value is "rbf." This is one of the most commonly used kernel functions, and the value of gamma influences how the data points are grouped together; low gamma means a larger similarity radius, which results in a greater number of points being grouped together. Thus, resulting in lesser chances of overfitting.

Testing and Evaluation of the model

After fitting our model, we use "model.predict()" to predict our result by passing first our validation data and then the test data. Before doing this, we have to transform the data for polynomial features and standardisation, not fit the data. We then generate the classification report and get a precision score that tells us the percentage of prediction, i.e., 86% for 0’s and 80% for 1’s for the test data. The F1 score tells us what percentage of the positive predictions were correct. We get 82% for 0’s and 90% for 1's, for an overall accuracy of 87%.

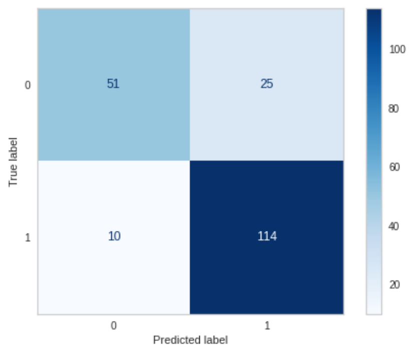

Figure 3: confusion matrix for SVM

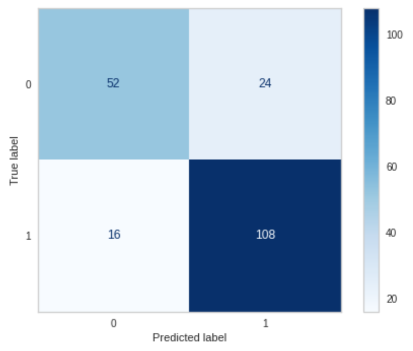

Figure 4 : confusion matrix for logistic regression

Figure 3 shows we had 114 correct classifications for 1s and 51 correct classifications for 0s. We also implemented the same classification using logistic regression and did not get a very good result. As we can see in Figure 4, the correct classification is less than that of SVM, and we get an accuracy of 80% on test data, which is less than that of SVM. Hence, we choose the SVM model over the logistic model.

5. Neural Networks

Data pre-processing:

Like the above two task we start off with pre-processing the data. Since our data contains categorical values, we apply the label encoding scheme to convert the textual categorical value to numeric values. We do this for the Region, Education, Work Type and Sex columns.

Since it’s a binary classification problem and we have to classify the data based on salary, i.e., either greater than 35000 or less than 35000, we introduce a column called "GreaterThan35k," in which we assign a value 1 if the salary is greater than 35k and 0 if the salary is less than 35k. Next, we select features and create our X and Y variables, where "X" is our feature matrix and "Y" is our target vector.

Splitting the data set

We split the data into a 60:20:20 split for training, testing, and validation and further standardise the data using StandardScaler() before building our model.

Fitting our model

We need to set a regularizer, we will be using a L2 regularizer from the tensorflow.keras package. This helps prevent overfitting in neural networks, hence improving the efficiency. Next, we create our layer for the network and by using the ‘Dense()’ function and specifying the kernel_regularizer. Since this is a classification problem, we make use of two layers.

Next, we set our model, i.e., the logistic regression model, and pass the parameter to the "Model()" function. We will be making use of the "Adam" optimizer. This is nothing but a function that changes the attributes of the network, for example, the weights and the learning rate, thus helping improve the accuracy. There are other optimizers that can be used, like "SGD," but after experimenting, the Adman optimizer proves to give a better result than "SGD."

We then initialise the BinaryCrossentropy() and the BinaryAccuracy() to measure the efficiency of our model. The binary cross accuracy compares the probability with a real-class output. It then calculates a score and penalises the probability depending on how far away the actual value is. whereas the binary accuracy measures how many predictions our model got right. Now we compile our model by running the "model.compile()" function, and we pass our optimizer, loss function, and evaluation matrix.

Testing and Evaluation

Now we run the "model.fit()" command and pass the training data, and after experimenting with different values, we set the "epochs" to 50 and the batch size to 100. "Epochs" are nothing but the number of complete passes our algorithm has had through the training data, and the batch size is the number of samples that will be propagated through the network.

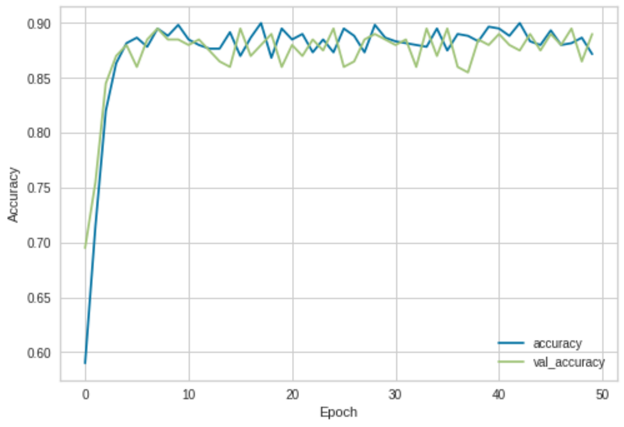

Figure 5 Graph of Accuracy vs Epoch

The graph in Figure 5 shows us the plot for the accuracy of training and validation data, and we can observe that the two lines don't deviate much from each other. Thus, we can safely say that we are not overfitting our data. We get an accuracy of 88% on test data. and an F1 score of 0.91. We can see in Figure 6 that we get 61 correct classifications for <35k and 118 correct classifications for >=35k.

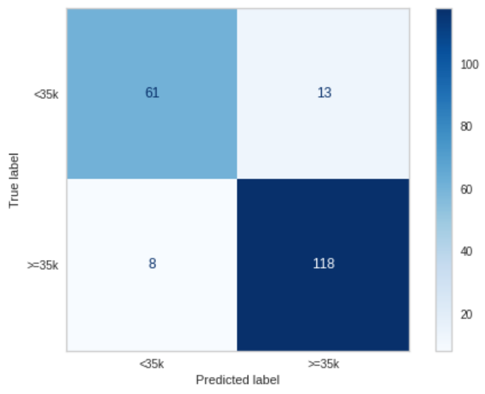

Figure 6:Confusion Matrix

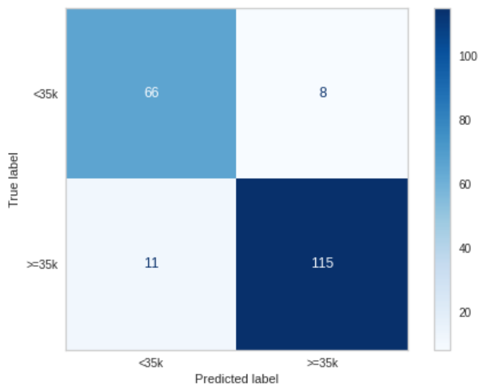

Figure 7: Confusion Matrix (class imbalance)

We further try to check for imbalances in our data and see if we can improve our results. We can do this by assigning weights to our class in the target data. This is done by first counting the 0s and 1s and using the below equation to calculate the weight.

```
class_weight = {0: m_total / (2.0 * m [0]), 1: m_total / (2.0 * m [1])}
```

We then train and test our model. We observe that the results do not have a big impact on the weight assigned to the class, but our model has definitely improved slightly. We get an accuracy of 90% and an F1 score of almost 92. In Figure 7, we can see that the classification numbers change, and the number of false positives and negatives reduces.

6. Clustering

Clustering is the process of dividing our data into a number of groups so that the data points within each group are more similar to one another and different from the data points within the other groups. It is essentially a grouping of items based on how similar or unlike they are to one another. We can draw references and useful information from our data set without using labelled responses.

To implement clustering on our data, we will be using the "KMeans" clustering algorithm from the "sklearn.cluster" package. First, we initialise a model using the KMeans() function and pass the number of clusters. To get the ideal number of clusters, we make use of the Silhouette method. Here, we compute the Silhouette score for cluster numbers k = 2 to k = 10. The Silhouette score is nothing more than a measure of how similar a data point is to other data points in the same cluster. This value ranges from +1 to -1.

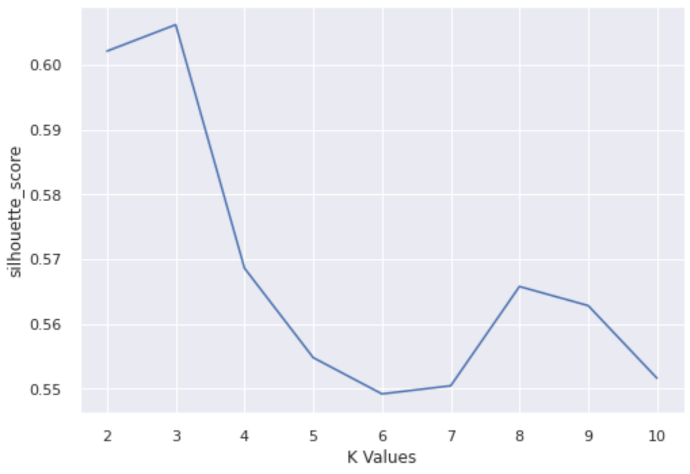

Figure 8: Ideal cluster number graph

From Figure 8, we can see that the ideal number of clusters is 3, as the silhouette score is the highest for this cluster, meaning the data points in our clusters will be similar to each other. Here, choosing the number of clusters as 3, which seems to be a good number, allows us to draw some useful insights after clustering. We chose the other attribute as "Age." Since we can draw some insights from our data with respect to age, we create a NumPy array of "age" and "salary" and use it to fit our model using the model.fit_predict() function. We can visualise our clustering results using the Seaborn library.

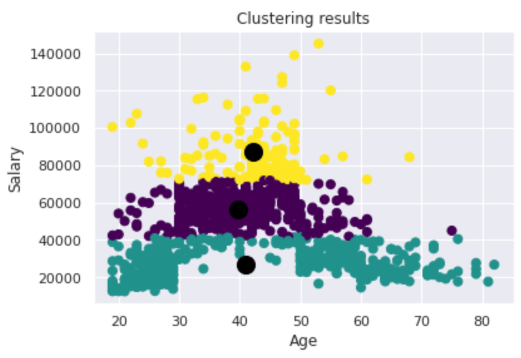

Figure 9:Clustering of salary based on age

According to Figure 9, the majority of people between the ages of 30 and 50 earn at least 35,000 per year. Whereas the purple region indicates that most of the people in the age group of 30 to 55 have a salary of 40,000 to 70,000. There are a lot of far scatter data points in the yellow region. These individuals may be outliers; they have high salaries, i.e., 80,000 and above, and are between the ages of 35 and 55.

7. Conclusion

We were provided with a data set that contained the information of customers of a company and the salary they make based on different attributes like "Age," "Education," "Work Type," "Site Time," "Region," and "Sex." By implementing different machine learning algorithms, we were able to predict the salary by making use of the attribute that influenced it.

In task 3, we used four different types of regression models, including Linear, Ridge, Lasso, and Random Forest. The Random Forest gives us the best R2 score of 0.8502, and since Ridge and Lasso are from the same family as Linear regression, we get similar R2 scores. In task 4, we are asked to perform Binary classification, which requires us to divide the data into two groups: customers with salaries greater than 35,000 and customers with salaries less than 35,000. We experimented with logistic regression and the SVM algorithms for classification and got a better score and an accuracy of 84% for the SVM, which was higher than Logistic regression. In task 5, we perform the same Binary classification using a Neural Network with Logistic regression and get an accuracy of 90% after performing class balancing. In task 5, we clustered the data into three groups and extracted insights from our visualisation. After implementing all of the above-mentioned algorithms, the Neural Network proved to have the best accuracy and, hence, is the best model for predicting salary.

8. References

Bartholomew, D. J., Knott, M., & Moustaki, I. (2011). *Latent variable models and factor analysis: A unified approach* (3rd ed.). Wiley-Blackwell.

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern Classification* (2nd ed.).John Wiley & Sons.

Hennig, R. (2015). *Handbook of Cluster Analysis* (C. Hennig, M. Meila, F. Murtagh, & R. Rocci, Eds.; 1st Edition). Chapman and Hall/CRC.

Ki, W. C. (2006). Gaussian processes for machine learning. *International Journal of Neural Systems*.