Machine Learning

Shanmugha Balan S V

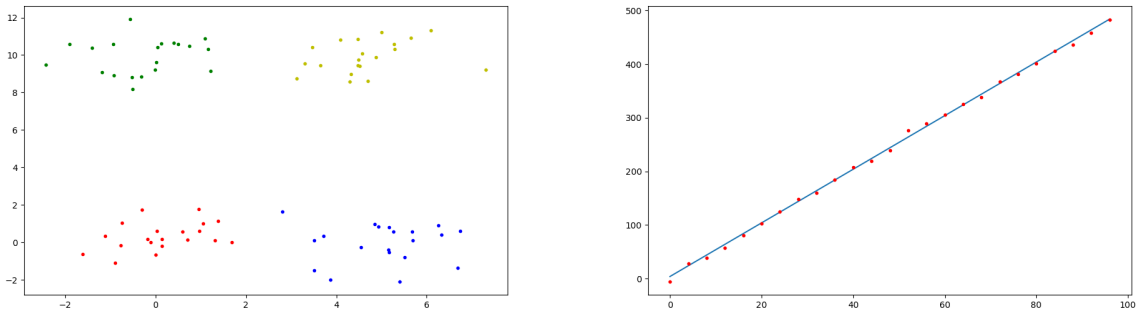
Contents

1 Machine Learning - Basics

1.1 Types of Learning

Machine Learning is broadly split into supervised learning and unsupervised learning. In supervised learning, we make the model fit to input-output pairs, so when the model is fed in with new data, it gives a similar output. The outputs are labelled in supervised learning. In unsupervised learning, the module makes arbitrary divisions, with no labelling of data. It finds clusters of similarities in the data presented to it.

Supervised learning methods mainly fall into two subcategories - classification and regression. In a classification problem, data is presented as belonging to a labelled group. It has discrete categories. Classification can be binary or multi class. Binary classification is between two groups - a positive class and a negative class. Multi class classification is when a data point should be placed in a group with many possible choices. Often, the problem is broken down into many binary classification problems. In regression, we have real number data, where we map an input to a single number output. They generally have some sort of continuity in the output or the data, and is often a matter of finding the best fit, by extrapolation or interpolation from the data.



(a) Classification (b) Regression

Figure 1: The two types of supervised learning

1.2 Choosing a Model

A machine learning model is evaluated on the basis of a loss function, which may be constructed differently for different problems. Naturally, we would want to select the model with the best accuracy or the least loss. There are two ways we can evaluate accuracy. We can check out how well the model predicts data we have already shown it, or training data. We can also show it brand new data or testing data. To test the model, often we split the data we have into training data and testing data by shuffling it and splitting. For this, we can use `sklearn.model_selection.train_test_split`. We now train the model on the training data and evaluate it on the testing data. We evaluate many different models on their losses and accuracies. But how do we choose a split of the data we have? To do this, we can use cross validation. Cross validation is when we split the data into folds, and evaluate the model one by one keeping a fold for testing, while training on the rest. We then use the cumulative results to decide which model to pick. Crossvalidation can also be used to find the optimal hyperparameters for our model. More on the implementation of cross validation is available in the scikit-learn docs.

1.3 Generalization of an ML Model

If our model simply memorizes the training data, we may not have good results as the outliers in the training data may induce weird biases to give awkward results during testing or when the model is deployed. So our model must learn sufficiently well from the data while still retaining a good ability to generalize to new data. If our model doesn't learn well enough from the training, it is said to underfit to the data. If our model does very well in training but poorly in testing, it is said to have overfit to the data. We need to find the right spot for the model

to ensure a balance between training and testing accuracies. Epochs are a training pass through the training data, where the model attempts to learn or adjust its weights and parameters. In the plot below, we can see the fitting of the model during training increases with epochs. However the testing accuracy increases to a point and then starts dropping. The phenomenon to the left of the testing maximum is called underfitting and the right side of the maximum is overfitting. For a trained model in scikit-learn, we can calculate the training accuracy as `model.score(X_train, y_train)` and the testing accuracy as `model.score(X_test, y_test)`. There are more functions for complex metrics in scikit-learn. You can also use predictive analysis with probabilities for models which support them.

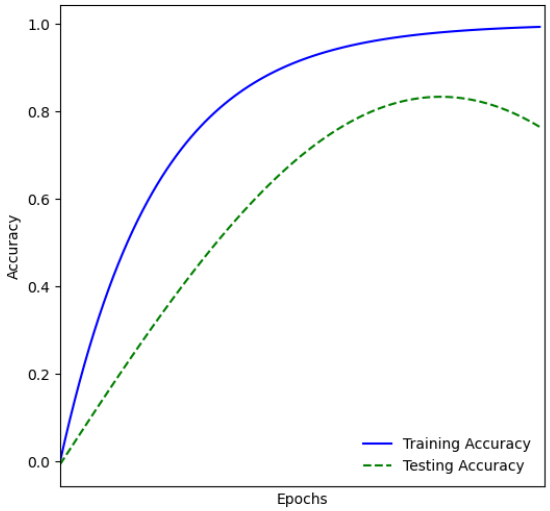
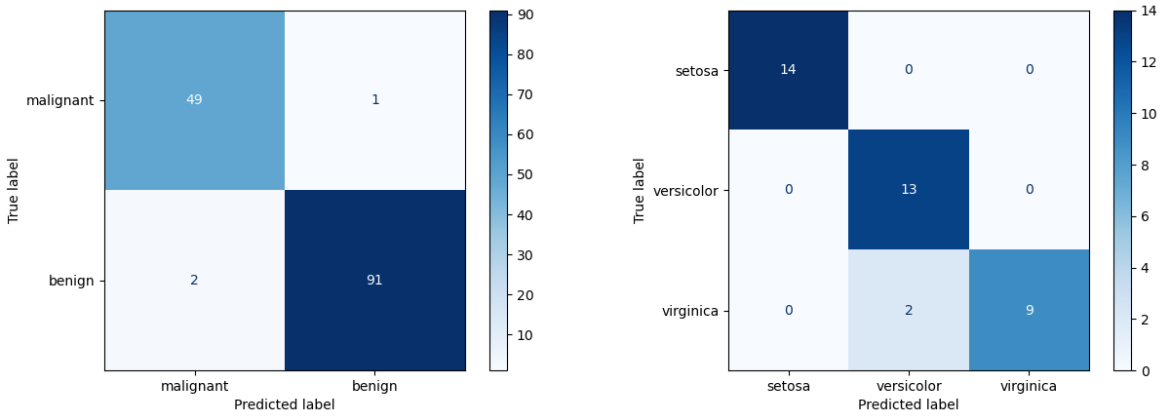


Figure 2: Training & Testing Fit for a Model

This also introduces two important terms in machine learning - bias and variance. Bias is the inability of a machine learning model to fit to the true data, in other words, it is the training error. The difference in fits between different sets is called variance. Even though the training data and the testing data come from the same underlying distribution, often they don't have the same accuracy. In relation with the previous terminology, when the model has a high bias, it is said to be underfitting the data, and when the model has a high variance, it is said to overfitting the data. We have to optimize and find the ideal model which has the least possible variance for the minimum bias.

1.4 Confusion Matrix

To analyse our model's successes and failures better, we make a confusion matrix. We can either use a normal binary two class confusion matrix, or even a multi class confusion matrix. With the example of the pre-existing datasets on the `sklearn` package, we load the iris and the breast cancer datasets. We train it with a supervised learning method - the polynomial support vector machine classifier and plot the correct classifications and the wrong classifications in a matrix. The true positives and true negatives (or correct classifications) are presented along the diagonal. Off diagonal results are the misclassfied erroneous vectors. If they are in the lower triangle of the matrix, it is a false positive, or a type I error. If they are in the upper triangle of the matrix, it is a false negative, or a type II error. Obviously, confusion matrices would be of size $n \times n$ for a n classification problem.



(a) Binary Classification (b) Multi Classification

Figure 3: Confusion matrices for the breast cancer and the iris dataset

1.5 Fancier Metrics

While the overall accuracy of the model can be calculated by summing the diagonal elements and dividing it by total elements, it often doesn't tell us the full picture. We can use the "sensitivity" or true positive rate or recall. It is the proportion of a class classified correctly. Similarly, we can define the "specificity" or the true negative rate. It is the proportion of wrong examples classified correctly, and would have more meaning for a binary classification problem. Precision is the ratio of correctly classified examples to all the examples classified so. As precision and recall are the most important metrics to draw from this, we use it to give one single number the F1 score, which is the harmonic mean of precision and recall.

$$F_1\ score = \frac{2 \times precision \times recall}{precision + recall}$$

All of these metrics are summarized in one table here.

		True condition			
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

Figure 4: The various metrics with which a Confusion Matrix can be analyzed

1.6 ROC Curves and AUC

Often, our problem influences the parameters of a model. If we are forced to reduce the amount of false positives, or false negatives, then our model accommodates them by compensating elsewhere. Reducing false positives often increases false negatives and vice versa. We can also use this analysis to simply determine the best parameters for our model. This relationship is mapped by a plot between the true positive rate (*sensitivity*) and false positive rate ($1 - specificity$). The plot is called receiver operator characteristic or ROC graph. For imbalanced data, it would be better to use precision instead of the false positive rate. The area under the curve is another metric to help us deciding the best model to pick. The model with the largest area under the ROC curve does the best.

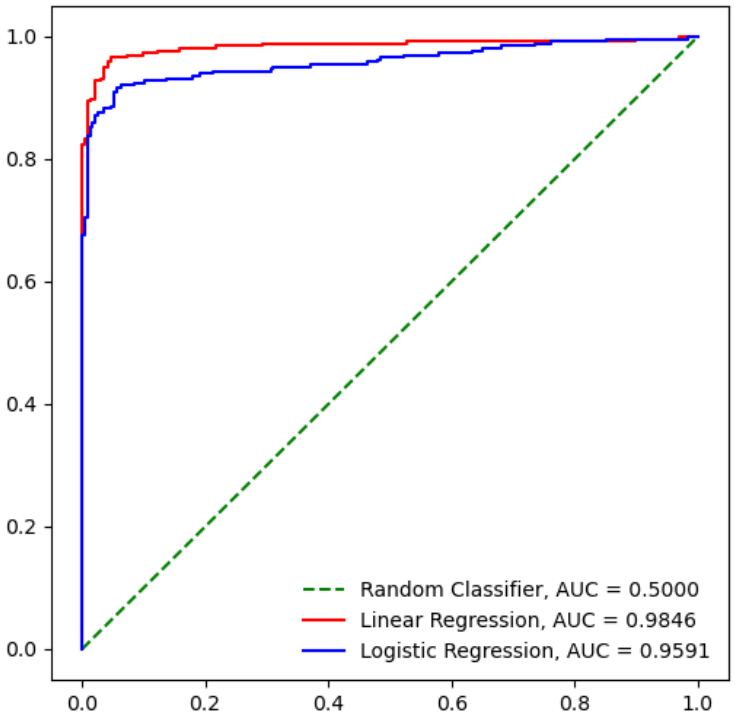


Figure 5: ROC Curve for two linear models

2 Supervised ML Algorithms

2.1 k-Nearest Neighbours

It is the simplest machine learning algorithm. It memorises the training set (yes). At test time, it iterates through the training set and finds the closest match to the vector presented to it. It classifies or regresses the new vector as belonging to the same class as the closest neighbour. This model has a hyperparameter k. Instead of immediately classifying the input vector as belonging to the nearest neighbour, the model now takes a vote between the k nearest neighbours for the vector. The downside of this model is that it takes $\mathcal{O}(n)$ time during prediction, while $\mathcal{O}(1)$ time during training. Usually, this is the other way round with machine learning models, where prediction is supposed to be quick, but training can take its time. Nonetheless, it is still put to use in many character recognition or OCR tasks, where training data is quite small, and the test input is very similar to the training data.

Scikit-learn provides the class `sklearn.neighbors.KNeighborsClassifier`, which takes in an argument k for the kNN algorithm. We train this on the iris dataset from `sklearn.datasets.load_iris`. We plot it for 6 values of k, namely, 1, 3, 7, 13, 21 and 51. Note how all values of k are odd - this is to reduce the possibility of a tie during the classification. Ties can be resolved by randomly picking a class. In the figures with low k, we see islands everywhere, and there are numerous overfitting artifacts as the model is very sensitive to outliers. Larger values of k smooth things out, but in an imbalanced dataset, the majority class might swamp the classification and affect performance.

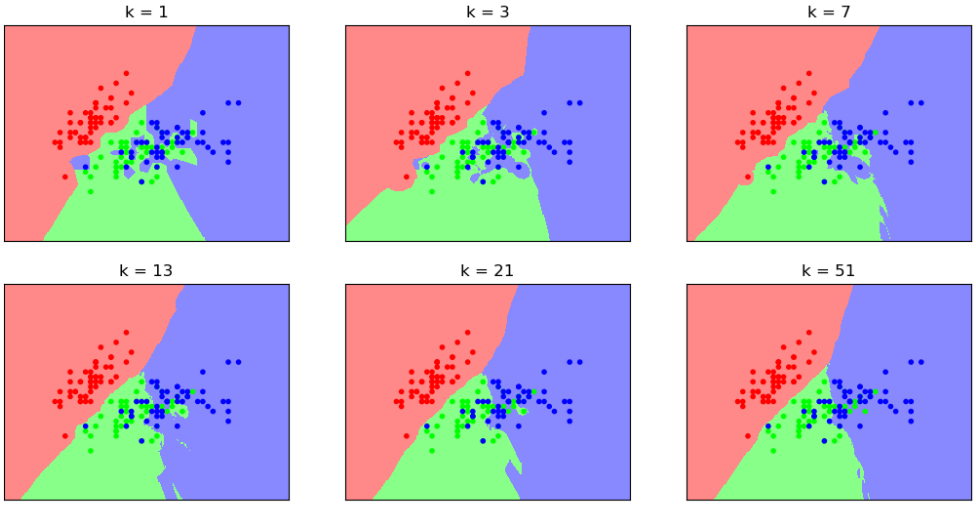


Figure 6: kNN Classification

More mathematically, the kNN algorithm takes the input vector v, and makes a pass through the training data. For every vector u_i in the training data, the norm of the difference is calculated as $l = ||u_i - v||$. The vectors u_i for which l is least is found out (by sorting or iterating or however), and the k lowest l values are pulled out separately. Among these k values, a vote of the classification data, $y_1, y_2, ..., y_n$, is taken. The majority class in these k vectors is found and the input vector is classified as $\hat{y} = y_{max}$.