The Comparative Analysis of Artificial Intelligence Image Classification

*Aiden Coffey*

**Overview:**

**1.1 The Oversight**

In our analysis, we've found that not all methods work equally well for every dataset. This article compares seven different ways of organizing data to see which ones are most accurate. In terms of f1-score, precision, AUC, and recall. For this article, we are going to be seeing which of the following seven algorithms work best for a basic AI image recognition database.

* Perceptron
* Linear Regression
* Linear SVM (Support Vector Machines)
* Non-Linear SVM (Support Vector Machines)
* Decision Tree
* Random Forest
* K-Nearest Neighbors

**1.2 Classification Expectations and Definitions**

**1.2.1 Classification Report Output**

A screenshot of a computer

Description automatically generated

**1.2.2 Understanding the Classification Report Output**

When looking at the classification report we can see that in Scikit Learn, it is broken down into 4 different major columns and shows the basis of the number of classifications in rows. But there is precision, recall, f1-score, and support that is being output for each row. Then there is an overall accuracy to be printed as well as a macro average and a weighted average to go with it. We were also looking for the AUC (Area Under the ROC (Receiver Operating Characteristic) Curve).

**1.2.3 Defining the Output**

**Precision** – Precision is measured based on the number of true positive predictions that are made amongst all instances tested. precision is used to find the accuracy of the total correct predictions. To find the Precision of a data set we use the formula P = TP / (TP + FP) … TP would be True Positives and FP would be False Positives.

**Recall** – Recall is the proportion of the computer's positive predictions among all real positive instances. The recall is fairly similar to the precision however, recall is used to detect the actual number of true instances throughout the data. The formula for calculating Recall is R = TP / (TP + FN) … TP is True Positives and FN is False Negatives.

**F1-Score** – The f1-score is the balance of the precision and the recall and finds the total number of incorrect instances between predictions and actuality in the datasets. To calculate f1 we would use the formula F1 = 2 \* (Precision \* Recall) / (Precision + Recall).

**Support** – Support represents the number of labeled occurrences for each class in the data set, so If you look at the example above and add together all the supports for each row. We will see that there is a total of 760. So that means there were 760 testing instances used in the model.

**Accuracy** –Accuracy is the overall *accuracy* or correctness of the model’s predictions based on all the True Positives, True Negatives, False Positives, and False Negatives. The way to calculate the accuracy would be A = (TP + TN) / (TP + TN + FP + FN).

**AUC –** The ROC curve is a plot of the rate of True Positives compared to the rate of False Positives. It roughly creates its own calculation of how accurate the accuracy of the TP and FP are. If the AUC is close to 1 then that means the AI will be able to properly detect what it is distinguishing between compared to if it were closer to 50 and just guessing.

**Macro Avg** – The Macro Average is the average of the metric that is calculated across all the different classes however there is no account for class imbalance, such as if there are more data in one group compared to another. The formula for Macro Average is MA = Σ Metric / Number of Classes … where the metric is the metric value for each class.

**Weighted Avg** – The weighted Average is the same as the Macro Average however there is an account for class imbalance. To calculate that data, we will use the formula WA = Σ (Metric \* Support) / Total Support … where support is the number of instances for each individual class.

**1.3 The Data**

We are using a singular data set which is 1797 data pieces with 10 different classifications. The possible options for classification are 0 to 9. The AI model will go through each image pixel by pixel and then decide as to what classification it thinks it is as a whole number to make a decision. Each image is broken into an 8 by 8 image creating 64 pixels for the model to go through for each testing and training sample.

A screenshot of a number

Description automatically generated

**1.4 The Computer Processing**

All of the following results have been done on an AMD Ryzen 5 5625U with Radeon Graphics and AMD Ryzen 5 5625U with Radeon Graphics. I used SciKit-Learn with Jupyter Notebooks in Python in version 3.9.18. For the data plots and graphs, we utilized the Mat Plot Libraries.

**Classification and Results:**

**2.1 Perceptron**

The perceptron, regarded as a foundational binary classification algorithm, mirrors the functionality of a single neuron within the brain. It acts as a single neuron in the brain. It will take input features and give them random weights to then calculate a sum and a bias then classify it into one of two classes.

When training a Perceptron you will use a test\_training\_split() algorithm to divide up the code and will input the data for your X and y values. The X value is the input features the Y is the corresponding labels. Then give it a size of data to test it on from the total X data, then give it a random shuffle to randomly select the data.

For my specific data set, I found that if I made the test size anything greater than 40%, I would receive a model that was not very accurate. The best data set that I was able to achieve was at a test size of 25% where I had an accuracy of 96%. With this, I decided to make the testing size of the rest of the data sets will be 25% of the training data as well for accurate representation.

3A graph of numbers and a line

Description automatically generated with medium confidence

4A screenshot of a computer screen

Description automatically generated

For the data, there were 18 missed classifications, and we can tell that there were no missed classifications for 0, 1, or 2, however, as we continued to move down the model had trouble depicting 8’s and was classifying them as 1’s most commonly. This data was just inputted to show that there was no change to the data and it was just a basic untouched perceptron model that can stack and compare to the rest of the data.

**2.2 Linear Regression**

Linear regression is an AI model that is used to show the relationship between one or more features and their target by creating a linear line to determine its official classification. When working with Linear regression you will have specific parameters that you can change to create a more optimal model.

You will have a parameter of C and when you set C to a value it just represents the strength regularization of the data. So, a smaller C value will get you a stronger regularization and vice versa. The solver is the next parameter, the solver is the way that the computer can optimize an algorithm that can be used to minimize the loss function. The multi-class is the way that the computer can handle the regression model for multiclass classification.

A graph of a graph with numbers and a line

Description automatically generated with medium confidence

A screenshot of a computer screen

Description automatically generated

For this method, the parameters that are in the image above are the ones that optimized my data to give me the best accuracy. I was able to get an accuracy of 96.888% with 450 total testing cases. 0 and 2 classes, both perfect in recall but in accuracies 2 was off by about 2 missed classifications.

**2.3 Linear SVM (Support Vector Machines)**

Linear SVM is a class of supervised learning models that are used for the classification of tasks. What they do is find the line for the linear regression that maximizes the distance between the different classes. That line must be linear which is why it is known as linear, but some parameters can be adjusted to optimize the model still.

The first one that I played with was the C value which was the same as in linear regression, the biggest thing was that it worked out exceptionally well for the model when I used an overly small C value. The one that I found worked the best was a c-value of 0.001. Then I set the class\_weight to balanced. The balance made it so that there was not much variation between the classes thus making it easier on the model. Then I had a dual set to false. When I was doing my research, it was recommended that with larger data sets it is faster overall on the model for the dual to be set to false. And since this data set is over 1000, I figured that was a larger enough data set. With this data set it was about the difference of 0.4 seconds but when you get even more data that could be the difference in hours.



A graph of a graph with numbers and a line

Description automatically generated with medium confidence

A screenshot of a computer screen

Description automatically generated

The Linear SVC model achieved perfect precision for class 2, demonstrating a notable accuracy of 96.296%. However, despite this relatively strong performance for one class, the overall recall metric did not exhibit exceptional results. In comparison to other classification methods applied to this dataset, Linear SVC did not show to be one of the top-performing models.

**2.4 Non-Linear SVM (Support Vector Machines)**

Non-Linear SVM is the same as SVM the only difference is that you can have the decision line be a circle, a curve, or anything that is not linear, the most notable difference is that to create the decision line, it takes the classified data plots, and turns it into a 3-D image and then finds the line that best splits it with largest gap between classifications, then it will take that position of the line and convert it back into a 2-D image which in turn will typically make it into a circular decision line.

For Non-Linear the parameters that I had set were the kernel at ‘rbf’ since that is what is used to specify that it is a Non-Linear SVM. I then set a C value and for this, it was the opposite a higher C value made it much more accurate, where the difference between a C value of 1 and 100 was about 7%. Then I had to set the gamma. The gamma for a Non-Linear training set determines the weight and importance of a singular training sample in the set. So, by making it ‘scale’ that means that there is an equal value in each training sample.



A graph of a number and a line

Description automatically generated with medium confidence

A screenshot of a computer screen

Description automatically generated

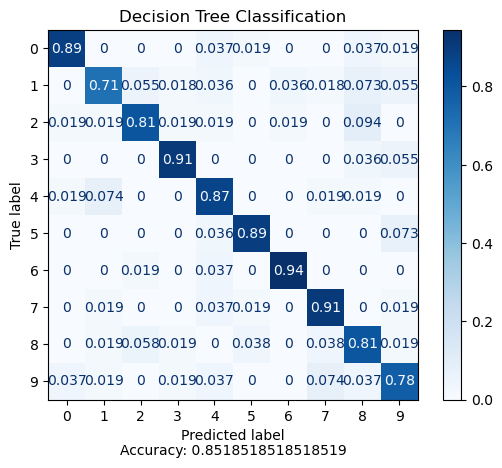
Overall, the Non-Linear model had one of the best outcomes, coming out at a near-perfect accuracy with only a few missed classifications. Non-linear in this form came out to be a much better alternative and typically can be found to work with a higher recall and f1-score on AI image recognition. That is not always the case, however it is fairly common.

**2.5 Decision Tree**

A Decision Tree is one of the most simplistic but useful forms of classification for data. It uses recursion to go through features and decide what the classification will be for the data. Using nodes to represent the decisions of the data and then using the whole leaf to tell what the final classification of the sample is.

For this data set, however, this was not the best choice to get a good accuracy overall. The parameters that I played around with were the max\_depth, min\_sample\_split, and min\_sample\_leaf. I found that those with the min that were set higher than 3 had about a 2% decrease with every number gone above 3 for this data set. And the max depth needed to be deep so 25 was a roughly good number. The set below was what I used to get my most optimal Accuracy which was the worst by far at about 85% precision accuracy.





A screenshot of a computer screen

Description automatically generated

The precision, recall, and F1 scores consistently hovering around the .7 mark indicate a suboptimal level of accuracy within the dataset. This renders it the poorest performer among all classification methods analyzed. While Decision Trees may occasionally emerge as the most suitable option, we can see there was an accuracy of 85.185%, which the current analysis suggests is not the case at present.

**2.6 Random Forest**

Random Forest is one of the more interesting AI models, it creates multiple different decision trees and then creates its own *forest* that then can be used to create a classification. These trees are completely randomized and are compiled into their own sub-set or accuracies to the specific classification. Then it compares all the trees and decides as a “*forest”* based on the mean accuracy and the mode number of classifications.

For this random forest, I played around with some of the parameters to see if there was a way to make it more efficient and I noticed that the default for the criterion was set to “gini” and that wasn’t the most optimal for this data set so when I changed to” entropy” I noticed a 2% increase in total accuracy which isn’t a whole lot when it is already at 95% however that bumped it up to 97% so that looked a little bit better. I also played around with the minimum number of splits per line and found that we went down almost 3% for every singular increase of a split. There was also a much lower accuracy if the max depth was less than 10 and greater than 25.

A graph of a number of numbers

Description automatically generated with medium confidence

A screenshot of a computer screen

Description automatically generated

For this data, it was 1 and 4 that were the only perfect classes out of 10 classes but overall, there seemed to be a very high accuracy. Achieving a notable accuracy rate of 97.777% and maintaining a minimum recall of 0.94, this method delivered satisfactory outcomes, placing it roughly in the middle of all the models.

**2.7 K-Nearest Neighbors**

The K-Nearest Neighbor method operates as a fairly straightforward principle: selecting a value and examining the classifications of the k-closest data points. The algorithm then assigns the classification based on the majority vote among these neighbors.

For this instance, I had a k value of around 6 and it worked very well, with 99.444% accuracy. I noticed that when we had a very high level of K there would be a very big skew in the data, however, there was nothing that was stopping it from selecting and since in the images, there was overall a significant number of black images.

The worst accuracy that I was able to produce was an 8.333% accuracy rating with a k value of 1347. Overall, when I changed some of the hidden parameters, I noticed that the settings that SciKit-Learn had pre-set were the best with the highest Accuracy overall. I changed the preset weights to see if there was a change in the AUC. I tried changing the algorithm, for the K-Nearest Neighbor the preset is auto but there was another that I found which was *ball-tree and kd\_tree,* however, it didn’t make the model any more accurate.

A diagram of a number of numbers

Description automatically generated

A screenshot of a computer screen

Description automatically generated

Overall K-Nearest Neighbor was the best to go with, having a near-perfect accuracy of 99.444% and only misclassifying a few 9’s. The recall showed there to be a lower near-perfect output, but if we look at the f1-score there was almost a straight 1.00 or 0.99 across the board excluding the 9’s classification.

**Conclusion:**

**3.1 The Data Analysis**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **AUC** | **Recall** | **Precision** | **F1-Score** | **Support** |
| **Perceptron** | **0.96** | **NA** | **0.96** | **0.96** | **0.96** | **450** |
| **Linear Regression** | **0.97** | **NA** | **0.97** | **0.97** | **0.97** | **450** |
| **Linear SVM** | **0.96** | **0.97** | **0.96** | **0.96** | **0.96** | **540** |
| **Non-Linear SVM** | **0.99** | **0.99** | **0.99** | **0.99** | **0.99** | **540** |
| **Decision Tree** | **0.85** | **0.92** | **0.84** | **0.85** | **0.85** | **540** |
| **Random Forest** | **0.98** | **0.92** | **0.98** | **0.98** | **0.98** | **540** |
| **K-Nearest Neighbor** | **0.99** | **0.995** | **0.99** | **0.99** | **0.99** | **540** |

**3.2 The Conclusion**

Overall, we can see that for this data set, there was plenty of variation. However, the ones that stood out the most were K-Nearest Neighbors for its incredibly high and consistent accuracy rating as well as having an AUC of .995. whereas on the other side, we saw that the decision tree had a major showing for the weaker side still having an accuracy of 85% which can be considered high, however, when it is compared to the rest of the data there was plenty of variation and significantly lower in terms of f1-score with the next closet being 0.11 higher. There is room to improve on this data set and overall, there should be more information to be provided and worked upon.

**3.3 Final Thoughts**

In my research, I began by collecting data from SciKit-Learn to explore the field of image detection AI. Through experimenting with different tests and adjusting parameters, I discovered that there's still so much untapped potential in this specific field. We're just scratching the surface of what's possible with image detection AI. There's a whole world of possibilities waiting to be explored.

I hope that my findings can serve as a starting point for others interested in this field. By sharing our knowledge and experiences, we can continue to push the boundaries of what's possible with image detection.

*Best of Luck with Further Development,*

*Aiden Coffey*

[*SciKit-Learn*](https://scikit-learn.org/stable/supervised_learning.html#supervised-learning)

[*GitHub*](https://github.com/AidenCoffey/AI-Image-Model)