

# MACHINE LEARNING

## Coursework Report

**Academic Year:** 2023/24

**Course:** BSc Computer Science

**Module:** Machine Learning

**Module Leader:**

**Student:** Nunzio Emanuele Sgroi

**Student ID:**

**Date Submission:** 15/12/2023

## Table of Contents

---

<b>Table of Figures</b> .....	2
<b>Week 3</b> .....	3
Task 1 .....	3
Task 2 .....	3
<b>Week 4</b> .....	5
Task 1 .....	5
Task 2 .....	6
Task 3 .....	9
<b>Week 5</b> .....	10
Task 1 .....	10
Task 2 .....	11
<b>Week 6</b> .....	13
Task 1 .....	13
<b>Week 7</b> .....	14
Task 1 .....	14
Task 2 .....	15
<b>Week 8</b> .....	18
Task 1 .....	18
Task 2 .....	18
<b>Week 9</b> .....	20
Task 1 .....	20
Task 2 .....	20
<b>Week 10</b> .....	22
Task 1 .....	22
Task 2 .....	23
<b>Week 11</b> .....	25
Task 1 .....	25
Task 2 .....	26
Task 3 .....	28
Task 4 .....	30
<b>References</b> .....	31

## Table of Figures

Figure 1 - Elbow Plot Indicating K=4 .....	4
Figure 2 - Four-Cluster Data Scatter Plot .....	4
Figure 3 – Week 4, Task 1: Confusion Matrix for Hospitalization Predictions.....	5
Figure 4 - Formulas for Calculating Model Performance Metrics.....	5
Figure 5 - Predictive Outcome Breakdown .....	5
Figure 6 - Calculated Performance Metrics .....	6
Figure 7 - Summary of Performance Metrics Results .....	6
Figure 8 - Python Code for Evaluating K-NN Classifier Performance with ROC Curve Analysis.....	6
Figure 9 - ROC Curve Comparison for K=3 and K=7 in K-NN Classification .....	7
Figure 10 - Confusion Matrices and Performance Metrics for K=3 and K=7 .....	8
Figure 11 - Formula to calculate F1 score.....	9
Figure 12 - Random Forest Classifier Output for Max Features = 2.....	10
Figure 13 - Random Forest Classifier Output for Max Features = 3.....	10
Figure 14 - Random Forest Classifier Output for Max Features = 4.....	11
Figure 15 – Week 5, Task 2: Decision Tree for Binary Classification Problem .....	11
Figure 16 - Formula to calculate Entropy.....	11
Figure 17 - Formula to calculate Information Gain.....	12
Figure 18 - Calculation of the Information Gain for A1 .....	12
Figure 19 - Calculation of the Information Gain for A2 .....	12
Figure 20 - Ground Truth vs. Perceptron Algorithm Classification with Decision Boundary .....	13
Figure 21 - Formula to calculate Misclassification Rate .....	13
Figure 22 - Plot Highlighting Misclassified Data in Perceptron Classification.....	13
Figure 23 - Comparative SVM Classifier Performance Across Different C Values .....	14
Figure 24 - SVM vs. Perceptron Misclassification Rates Comparison.....	15
Figure 25 - Week 7 Task 2: Training Data for Linear SVM.....	15
Figure 26 - SVM Decision Boundary.....	16
Figure 27- Cross-Validation Accuracy Scores for Selecting Optimal k in k-NN .....	20
Figure 28 - Image Classifier tested on half CIFAR10 training data.....	21
Figure 29 - Week 10, Task 1: Convolution Filter Operation.....	22
Figure 30 - 3x3 Convolution Filter Example for Calculation.....	22
Figure 31 - Convolution Operation Calculation .....	22
Figure 32 - Training Accuracy vs. Validation Accuracy Across Epochs.....	23
Figure 33 - Epoch-to-Accuracy Comparisons Before and After Adjusting CNN Parameters .....	24
Figure 34 - Week11, Task 1: Regression Model Data .....	25
Figure 35 - Formula to calculate LSE .....	25
Figure 36 - Example Calculation of LSE .....	25
Figure 37 - Python code that compute LSE.....	26
Figure 38 - Week 11, Task 2: Regression Model Data.....	26
Figure 39 - Python Code that compute LSE and Identifies Optimal Model with lowest LSE .....	27
Figure 40 - Python Code with results for Week 11, Task 3 .....	28
Figure 41 - Python Plot of Sales Regression Based on Marketing Spend .....	29
Figure 42 - Formula to calculate Cost Function J .....	30
Figure 43 - Python Code that Compute Cost Function J .....	30

## Week 3

---

### Task 1

**Q:** Describe the issues associated with K-means algorithm, and explain the mitigating options.

K-means, a popular clustering algorithm, partitions data into K distinct clusters. It iteratively assigns data points to the nearest centroid and recalculates centroids until convergence. However, several issues can impact its effectiveness:

- 1. Incorrect Number of Clusters (K):**
  - **Issue:** Choosing the wrong K can lead to ineffective clustering.
  - **Mitigation:** Using the Elbow method, where the optimal K is indicated by a bend in the plot of variance explained versus the number of clusters. Alternatively, Hierarchical Clustering helps visualize the optimal number of clusters through a dendrogram, although it's computationally intensive for large datasets.
- 2. Sensitivity to Initial Centroids:**
  - **Issue:** Initial centroid placement can significantly affect the final clusters.
  - **Mitigation:** Multiple runs with different initializations can help, but a more efficient approach is the K-means++ algorithm, which optimizes initial centroid selection (Kumar, 2020).
- 3. Vulnerability to Outliers:**
  - **Issue:** Outliers can skew centroid calculations, affecting cluster quality.
  - **Mitigation:** Pre-processing data to remove outliers is one approach, but in cases where outliers are relevant to clustering, algorithms like K-medoids that are less sensitive to outliers can be considered (Google, 2022).
- 4. Non-convex Clusters Limitation:**
  - **Issue:** K-means tends to form spherical clusters and may not work well with non-convex shapes.
  - **Mitigation:** Techniques like Spectral Clustering can be used for identifying complex cluster shapes (Google, 2022).
- 5. Dependency on Feature Scaling:**
  - **Issue:** The algorithm is sensitive to the scale of data features.
  - **Mitigation:** Standardizing or normalizing data features ensures a fair influence on the clustering process (Roy, 2020).

In conclusion, while K-means is a widely used clustering method, it's crucial to be aware of its limitations and apply appropriate mitigation strategies to ensure effective clustering outcomes.

### Task 2

**Q:** The dataset we are going to use has 1000 entries with unknown number of clusters (download data\_Kclusters.txt from Blackboard). Use the Elbow Method to find the optimum value for K and show the results for clustering using the optimum k.

In the application of the Elbow method to the given dataset, the analysis identified the optimal number of clusters (K) as 4. This conclusion was reached through trials with different ranges for K: initially from 1 to 10 ( $K = \text{range}(1, 10)$ ) and subsequently from 2 to 10 ( $K = \text{range}(2, 10)$ ), with both ranges reaching the same result. The graph below illustrates the Elbow method's outcome, clearly

marking  $K = 4$  as the point where the rate of decrease in within-cluster sum of squares (WCSS) notably diminishes.

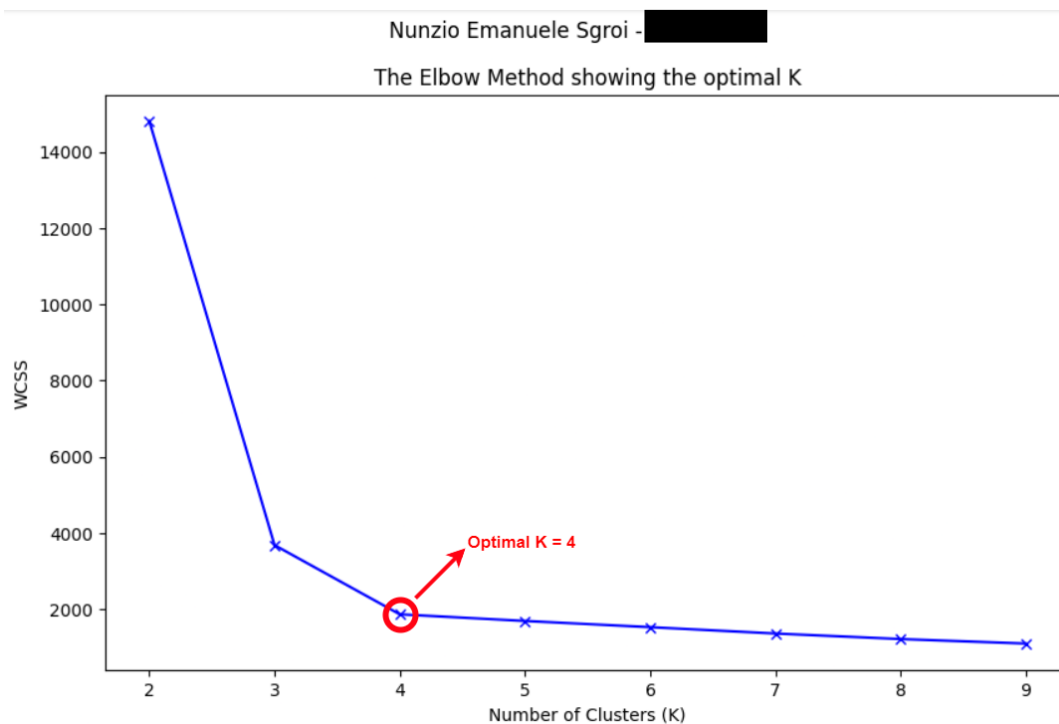


Figure 1 - Elbow Plot Indicating K=4

Following the identification of the optimal cluster count (K), the K-means algorithm was applied to categorize the dataset. In this process, each data point was classified into one of the clusters, with the assignment being guided by the closeness of the point to the corresponding cluster centroid.



Figure 2 - Four-Cluster Data Scatter Plot

## Week 4

### Task 1

**Q:** Let us assume we developed a machine learning model to predict which patients on dialysis will be admitted to the hospital in the next week. Given 100 patients, we have the following break-down:

- *True Positives: people that are hospitalized that you predict will be hospitalized*
- *True Negatives: people that are NOT hospitalized that you predict will NOT be hospitalized*
- *False Positives: people that are NOT hospitalized that you predict will be hospitalized*
- *False Negatives: people that are hospitalized that you predict will NOT be hospitalized*

Calculate the performance metrics in the table below:

Confusion Matrix		Actual	
		Hospitalized	Not Hospitalized
Predicted	Hospitalized	33	10
	Not Hospitalized	17	40

Recall (Sensitivity)	
Specificity	
Accuracy	
Precision	


  


Figure 3 – Week 4, Task 1: Confusion Matrix for Hospitalization Predictions

To calculate the performance metrics based on the confusion matrix provided, I used the following formulas:

$$\text{Recall (Sensitivity)} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

$$\text{Accuracy} = \frac{TP + TN}{\text{Total Cases}}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

Figure 4 - Formulas for Calculating Model Performance Metrics

The number reported by the confusion matrix, needed for the calculation, are:

- **True Positives (TP):** 33
- **True Negatives (TN):** 40
- **False Positives (FP):** 10
- **False Negatives (FN):** 17
- **Total Cases:** 100



Figure 5 - Predictive Outcome Breakdown

Therefore, the calculation of each metric is:

$$\text{Recall (Sensitivity)} = \frac{TP}{TP + FN} = \frac{33}{(33 + 17)} = \frac{33}{50} = 0.66$$

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{40}{40 + 10} = \frac{40}{50} = 0.80$$

$$\text{Accuracy} = \frac{TP + TN}{\text{Total Cases}} = \frac{33 + 40}{100} = \frac{73}{100} = 0.73$$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{33}{33 + 10} = \frac{33}{43} = 0.767$$

Figure 6 - Calculated Performance Metrics

The table can be filled as follow:

<b>Recall (Sensitivity):</b>	0.66 (or 66%)
<b>Specificity</b>	0.80 (or 80%)
<b>Accuracy</b>	0.73 (or 73%)
<b>Precision</b>	0.767 (or 76.7%)

Figure 7 - Summary of Performance Metrics Results

## Task 2

**Q:** Try K-NN classifier for dataset provided in for different numbers of neighbours (K = 3 and K = 7). Which K provides better results? Justify your answer by using AUC of ROC curve. Also provide the relevant performance metrics such sensitivity, etc.

To provide a clearer answer, I updated the Python code so it can check two different settings (K = 3 and K = 7) with a single run, making it easier to understand by showing the comparison on the same ROC curve graph. The code iterates for each K value, executing the requisite computations for the performance metrics and generating the ROC curve.

```
[23] #Nunzio Emanuele Sgroi - ██████████

def evaluate_model(n_neighbors):
    # Initialize and train model
    model = KNeighborsClassifier(n_neighbors=n_neighbors)
    model.fit(trainX, trainy)

    # Predictions
    y_pred = model.predict(testX)
    y_proba = model.predict_proba(testX)[:, 1]

    # Calculate metrics
    metrics = {
        'confusion_matrix': confusion_matrix(testy, y_pred),
        'accuracy': accuracy_score(testy, y_pred),
        'precision': precision_score(testy, y_pred),
        'recall': recall_score(testy, y_pred),
        'f1_score': f1_score(testy, y_pred),
        'auc': roc_auc_score(testy, y_proba)
    }

    # Calculate and plot ROC curve
    fpr, tpr, _ = roc_curve(testy, y_proba)
    plt.plot(fpr, tpr, label=f'K={n_neighbors} (AUC = {metrics["auc"]:.2f})')

    return metrics

results = {}
neighbors = [3, 7]

# Evaluate models and collect results
for n in neighbors:
    results[n] = evaluate_model(n)

# Plot the ROC curve
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.suptitle("Nunzio Emanuele Sgroi - ██████████")
plt.legend(loc="lower right")
plt.show()
```

Figure 8 - Python Code for Evaluating K-NN Classifier Performance with ROC Curve Analysis

Student: Nunzio Emanuele Sgroi  
Student ID:

The two cells of code, generate a graphical representation of the ROC curve:

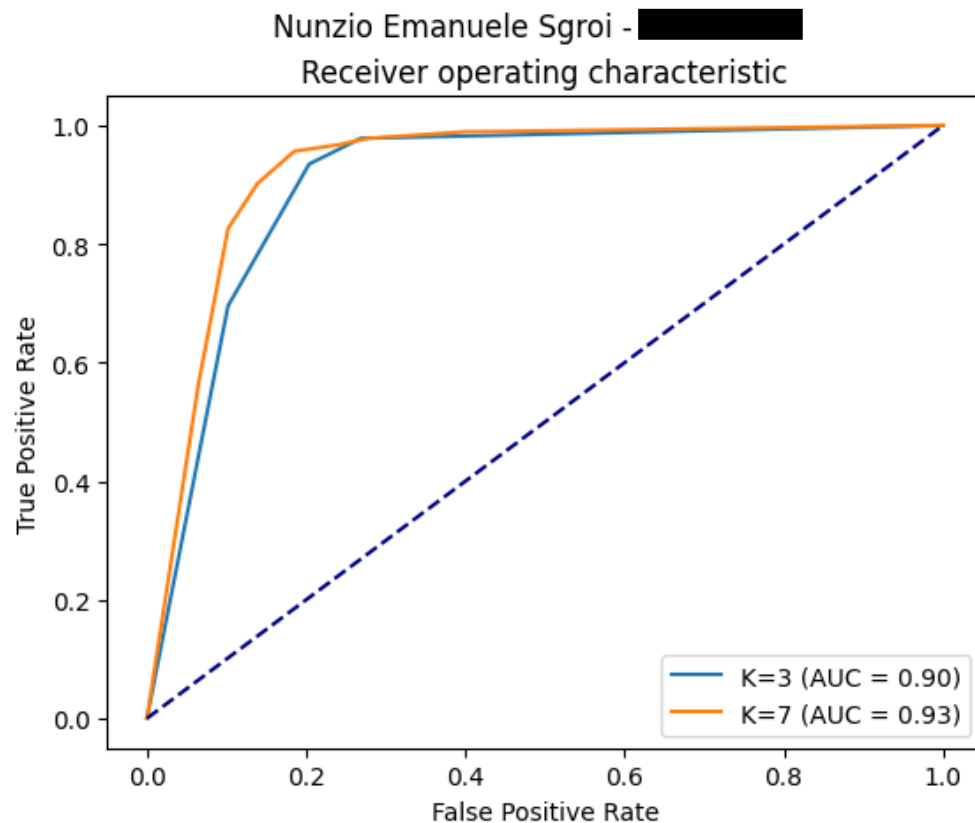


Figure 9 - ROC Curve Comparison for K=3 and K=7 in K-NN Classification

### ROC Curve Analysis

- For K=3, the AUC is **0.90**.
- For K=7, the AUC is **0.93**.

The AUC represents the ability of the classifier to distinguish between the two classes. An AUC of 1.0 represents a perfect classifier, while an AUC of 0.5 represents a worthless classifier. In this case, the classifier with K=7 has a higher AUC, indicating that it has a better overall performance in terms of distinguishing between the classes.

### Performance Metrics

The calculated performance metrics and confusion matrix for the two neighbour values (K = 3 and K = 7) are presented below:

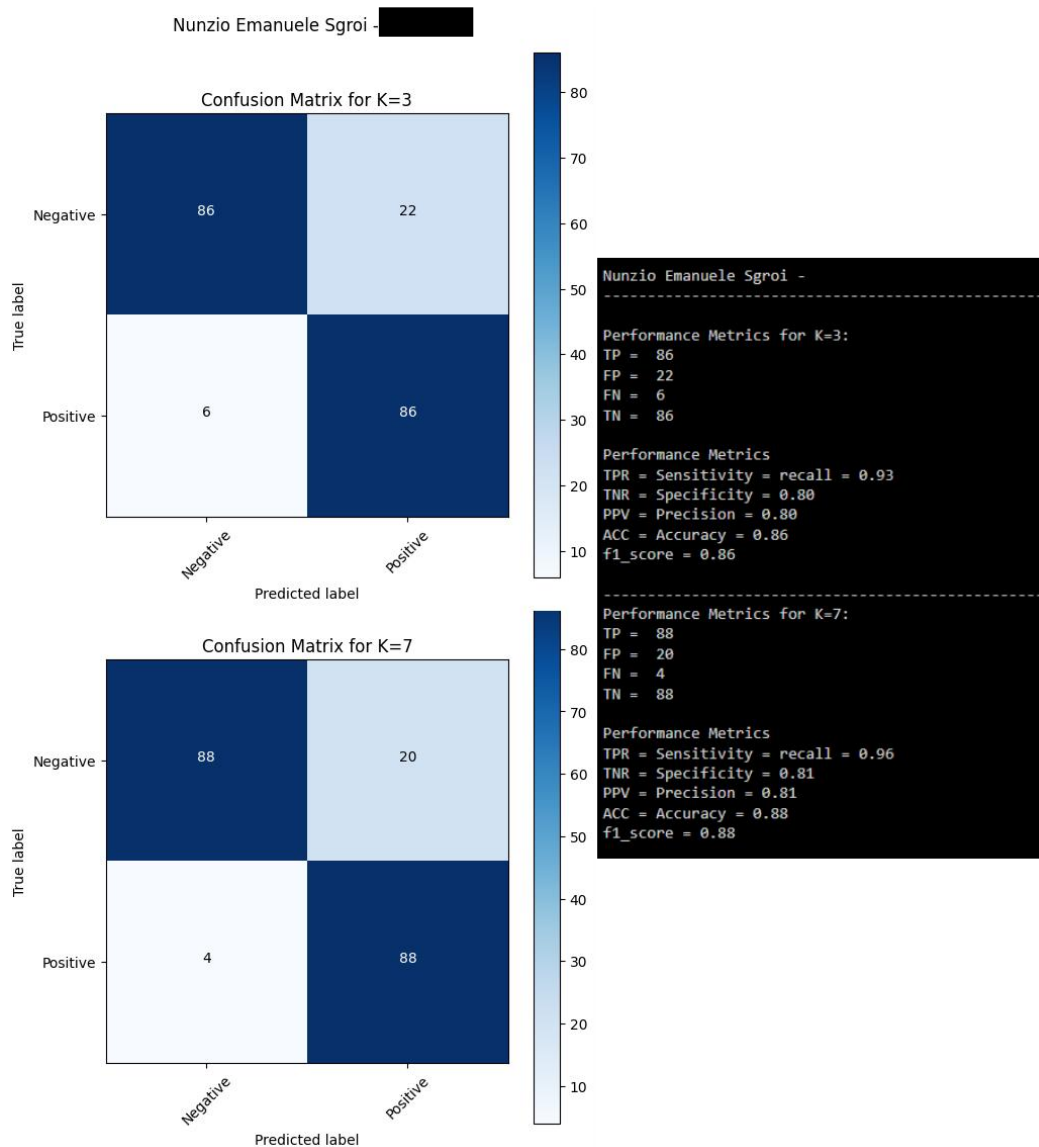


Figure 10 - Confusion Matrices and Performance Metrics for K=3 and K=7

- Sensitivity is higher for K=7, meaning it correctly identifies positives more frequently.
- Specificity is slightly better for K=7, indicating improved identification of negatives.
- Precision is the same for both K values, showing no change in the positive predictive value.
- Accuracy is higher for K=7, indicating a greater proportion of correct predictions.

In conclusion, the K-NN classifier with K=7 provides better results. This is supported by a higher AUC value, which signifies a greater ability to differentiate between classes. Additionally, all other performance metrics (sensitivity, specificity, precision and accuracy) are equal to or better for K=7 than for K=3. It's important to note that while K=7 is better in this instance, the optimal value of K can vary depending on the dataset and the balance between bias and variance. A larger K generally means less variance and more bias, whereas a smaller K can lead to more variance and less bias (IBM, no data).

**Note:** “f1\_score” was not mentioned in the paragraph above, as it is part of the Task 3.

### Task 3

**Q:** *The script also returns a value called 'f1\_score'. Do some research to understand what this measure is. Repeat task-2 and this time justify your answer using f1\_score. Do you arrive at the same conclusion?*

The F1 score is a measure of a test's accuracy. It considers both the precision  $p$  and the recall  $r$  of the test to compute the score:  $p$  is the number of correct positive results divided by the number of all positive results returned by the classifier, and  $r$  is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive) (Kundu, 2022).

The formula to calculate the F1 score is:

$$f1\_score = 2 * \left( \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}} \right)$$

Figure 11 - Formula to calculate F1 score

The best value of an F1 score is 1 (perfect precision and recall), and the worst is 0. It's a good way to show that a class has a balance between precision and recall. It is especially useful when the class distribution is uneven.

The f1\_score outputs for K=3 and K=7, reported in Task 2, are:

- For K=3: f1\_score = **0.86**.
- For K=7: f1\_score = **0.88**.

Comparing the two values of K using the F1 score, K=7 yields a slightly higher F1 score than K=3. This indicates a better balance between precision and recall for K=7, suggesting that this K value may offer a more accurate model for this particular dataset, aligning with conclusions drawn from the AUC of the ROC curve.

## Week 5

### Task 1

**Q:** Try Random Forest Classifier for the Iris dataset example, and when creating decision trees from the bootstrapped dataset, use different number of features:

- a subset of 2 features
- a subset of 3 features
- a subset of 4 features

And compare the accuracy of your Random Forest Classifier. Which one is better? Discuss your observations and justify your answer.

What is the largest number of features you may have?

Testing and comparing the accuracy of the Random Forest Classifier with different subsets of features (2, 3, and 4 features), produced the following result:

#### Max Features = 2

The classifier correctly identified all instances of the Setosa species, while it showed some misclassification between Versicolor and Virginica. The overall accuracy achieved was **0.9375**.

```
Nunzio Emanuele Sgroi - [REDACTED]
----> clf = RandomForestClassifier(max_features=2, random_state=0)

Predicted Species  setosa  versicolor  virginica
Actual Species
setosa             13         0           0
versicolor         0          5           2
virginica          0          0          12
Accuracy: 0.9375
```

Figure 12 - Random Forest Classifier Output for Max Features = 2

#### Max Features = 3

With an additional feature, the classifier maintained perfect classification of Setosa and improved in distinguishing Versicolor from Virginica, leading to a higher overall accuracy of **0.96875**.

```
Nunzio Emanuele Sgroi - [REDACTED]
----> clf = RandomForestClassifier(max_features=3, random_state=0)

Predicted Species  setosa  versicolor  virginica
Actual Species
setosa             13         0           0
versicolor         0          6           1
virginica          0          0          12
Accuracy: 0.96875
```

Figure 13 - Random Forest Classifier Output for Max Features = 3

### Max Features = 4

With 4 features in the dataset, the classifier's performance mirrored the results of the 2-feature model, with an accuracy of **0.9375**.

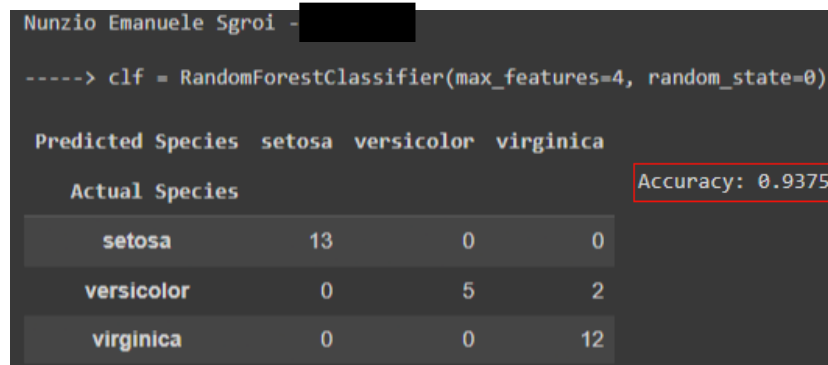


Figure 14 - Random Forest Classifier Output for Max Features = 4

When using a subset of **3** features, the classifier achieved the highest accuracy, which means it is the most effective. The perfect classification of Setosa across all models indicated that this species was easily distinguishable from the others. However, the crucial factor in differentiating Versicolor and Virginica lay in the number of features used, with 3 features providing the best results.

For the Iris dataset, **4** is the largest number of features we can have, however, as reported above, a subset of **3** features resulted more beneficial.

### Task 2

**Q:** You have been given a binary classification problem (positive/negative) where the original dataset contains 29 positive and 35 negative samples. We have 2 features of A1 and A2 which can be used for splitting the data. We would like to build a decision tree and figure below provides the resulting splits for each feature. Which feature would you use to split the data? A1 or A2? Justify your answer.

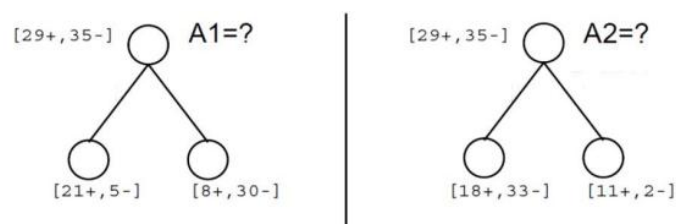


Figure 15 – Week 5, Task 2: Decision Tree for Binary Classification Problem

To understand which feature is more suitable to split data between A1 and A2, it is necessary to calculate the information gain (IG) for each feature. IG is based on the concept of “Entropy”, which measures the impurity or uncertainty in a group of examples.

To calculate the Entropy, I used the following formula:

$$Entropy(S) = -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Figure 16 - Formula to calculate Entropy

Student: Nunzio Emanuele Sgroi  
Student ID:

Where,  $S$  is a sample of training examples,  $p_{\oplus}$  is the probability of positive examples in  $S$ , and  $p_{\ominus}$  is the probability of negative examples in  $S$ .

To calculate the IG, I used the following formula:

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Figure 17 - Formula to calculate Information Gain

Where,  $S_v$  is entropy of each subset, weighted by fraction of examples  $|S_v|/|S|$  that belong to  $S_v$ .

#### Calculation of the Information Gain for A1:

$$Entropy([29+, 35 -]) = -\left(\frac{29}{64}\right) \log_2 \left(\frac{29}{64}\right) - \left(\frac{35}{64}\right) \log_2 \left(\frac{35}{64}\right) = -0.453 \times (-1.142) - 0.547 \times (-0.870) = \mathbf{0.994}$$

$$Entropy([21+, 5 -]) = -\left(\frac{21}{26}\right) \log_2 \left(\frac{21}{26}\right) - \left(\frac{5}{26}\right) \log_2 \left(\frac{5}{26}\right) = -0.807 \times (-0.308) - 0.192 \times (-2.378) = \mathbf{0.706}$$

$$Entropy([8+, 30 -]) = -\left(\frac{8}{38}\right) \log_2 \left(\frac{8}{38}\right) - \left(\frac{30}{38}\right) \log_2 \left(\frac{30}{38}\right) = -0.210 \times (-2.248) - 0.789 \times (-0.341) = \mathbf{0.742}$$

$$Gain(S, A1) = 0.994 - \left(\frac{26}{64}\right) \times 0.706 - \left(\frac{38}{64}\right) \times 0.742 = \mathbf{0.266}$$

Figure 18 - Calculation of the Information Gain for A1

#### Calculation of the Information Gain for A2:

$$Entropy([29+, 35 -]) = -\left(\frac{29}{64}\right) \log_2 \left(\frac{29}{64}\right) - \left(\frac{35}{64}\right) \log_2 \left(\frac{35}{64}\right) = -0.453 \times (-1.142) - 0.547 \times (-0.870) = \mathbf{0.994}$$

$$Entropy([18+, 33 -]) = -\left(\frac{18}{51}\right) \log_2 \left(\frac{18}{51}\right) - \left(\frac{33}{51}\right) \log_2 \left(\frac{33}{51}\right) = -0.353 \times (-1.502) - 0.647 \times (-0.628) = \mathbf{0.937}$$

$$Entropy([11+, 2 -]) = -\left(\frac{11}{13}\right) \log_2 \left(\frac{11}{13}\right) - \left(\frac{2}{13}\right) \log_2 \left(\frac{2}{13}\right) = -0.846 \times (-0.241) - 0.154 \times (-2.700) = \mathbf{0.619}$$

$$Gain(S, A2) = 0.994 - \left(\frac{51}{64}\right) \times 0.937 - \left(\frac{13}{64}\right) \times 0.619 = \mathbf{0.121}$$

Figure 19 - Calculation of the Information Gain for A2

In decision tree learning, the goal is to choose the split that results in the largest information gain, which means that the split adds the most information about the target variable by reducing uncertainty the most. Since the Information Gain for A1 (0.266) is greater than the Information Gain for A2 (0.121), feature A1 is the better choice for splitting the data. It will provide a more significant reduction in entropy compared to A2.

## Week 6

### Task 1

**Q:** Apply the Perceptron algorithm to obtain a linear classifier and plot the results.

- Is the data linearly separable? Justify your answer.
- If linearly inseparable, what is the percentage of misclassified points using your linear classifier??

Applying the Perceptron algorithm to the dataset "data\_Perceptron.txt" has shown that the data is **not linearly separable**. This conclusion is drawn from the visualization of the classification results, where the decision boundary, illustrated by a hyperplane, does not entirely separate the two classes. As evident in the image below, several points from both clusters are on the wrong side of the hyperplane, indicating misclassification.

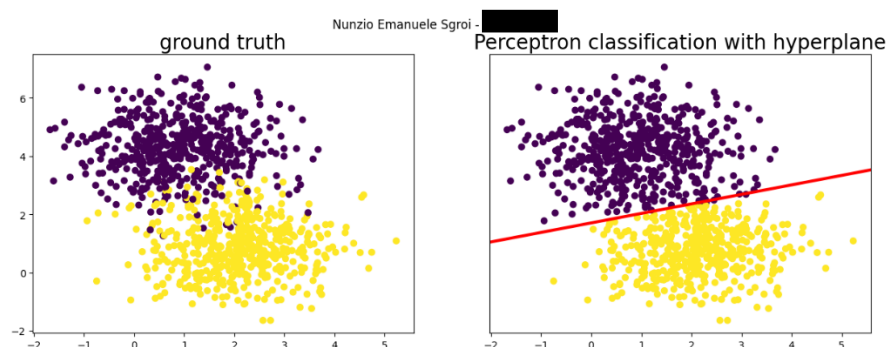


Figure 20 - Ground Truth vs. Perceptron Algorithm Classification with Decision Boundary

The misclassification rate is quantified by the proportion of incorrectly classified points out of the total number of points, multiplied by 100%. By running the code, it was determined that the total number of misclassified points is **56**. Given that the dataset contains 1000 points, the resulting misclassification rate is **5.60%**.

$$\text{Misclassification Rate} = \frac{\text{Total Misclassified Points}}{\text{Total Number of Points}} \times 100\% = \frac{56}{1000} \times 100\% = 5.60\%$$

Figure 21 - Formula to calculate Misclassification Rate

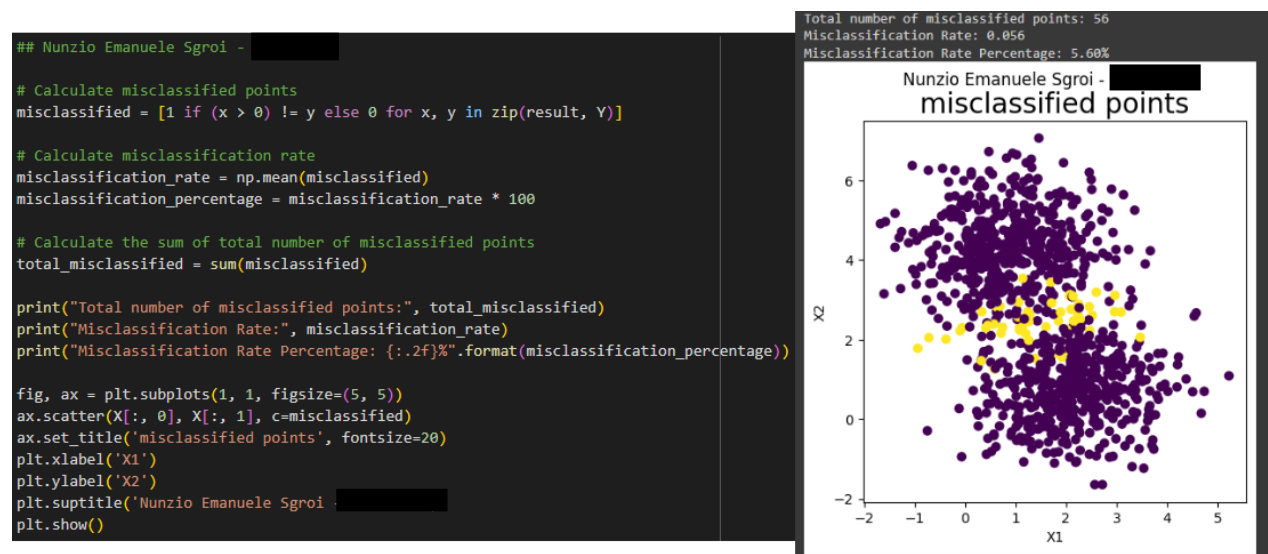


Figure 22 - Plot Highlighting Misclassified Data in Perceptron Classification

Student: Nunzio Emanuele Sgroi  
Student ID:

## Week 7

### Task 1

**Q:** Apply the SVM algorithm to obtain a classifier and plot the results.

- What is the percentage of misclassified points using your SVM classifier?
- Try the algorithm for a range of different values of C parameter [0.01, 0.1, 1.0], and provide the results together with your understanding of the influence of the C parameter.
- Compare your SVM results with those from the Perceptron in week-6.

The application of the Support Vector Machine (SVM) algorithm on the given dataset with different C parameters (0.01, 0.1, 1.0) generated the following results:

- **C = 0.01:** In this setting, the SVM classifier showed a **4.5%** rate of incorrect classifications. With a lower C value, which indicates a preference for a wider margin between classes, there is a balance aimed at correctly classifying most of the training points while keeping the model simple. The fairly low rate of misclassification in this case suggests that the model generalizes well, even with a simpler boundary for decision-making.
- **C = 0.1:** Raising the C value to 0.1 slightly lowered the rate of incorrect classifications to **4.3%**. This small change suggests that in this range of C values, the complexity of the decision boundary does not significantly affect how well the classifier performs. It indicates that the data is reasonably well-separated even with a boundary of moderate complexity.
- **C = 1.0:** Further increasing C to 1.0 showed a slight improvement in accuracy, with the misclassification rate dropping to **4.2%**. This trend suggests that a tighter margin, aimed at reducing classification errors, slightly enhances the model's accuracy.

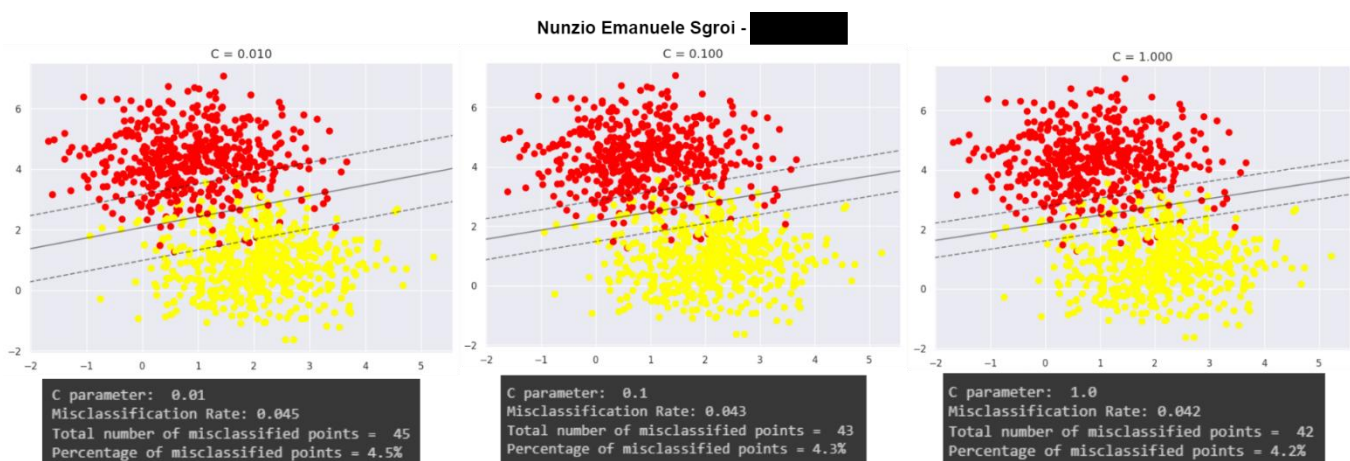


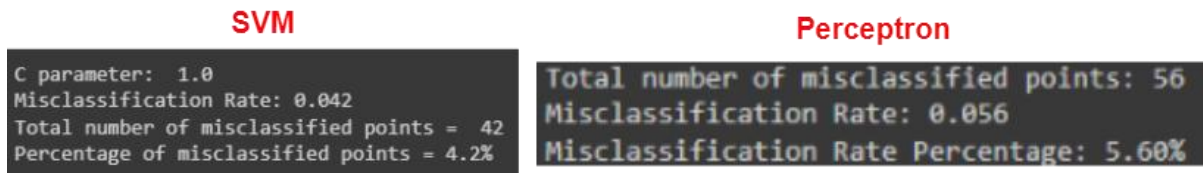
Figure 23 - Comparative SVM Classifier Performance Across Different C Values

In the context of this dataset, increasing the C value in the SVM algorithm generally led to more accurate classifications, suggesting that a stricter boundary for deciding between classes can be beneficial. This highlights the importance of adjusting hyperparameters such as C with attention to both reducing errors and avoiding the risk of the model becoming too narrowly focused on the training data, a problem known as overfitting.

Comparing the results obtained from the application of the SVM algorithm with those from the Perceptron algorithm employed in week 6, based on the same dataset, it was observed that the

Student: Nunzio Emanuele Sgroi  
Student ID:

Perceptron algorithm has a higher misclassification rate of **5.60%**. This comparison reveals that for this specific dataset, SVM with a suitably selected C parameter is more effective than the simpler Perceptron model. The lower misclassification rates in SVM highlight its capability in establishing an optimal separation plane, particularly beneficial for datasets with distinct class divisions.

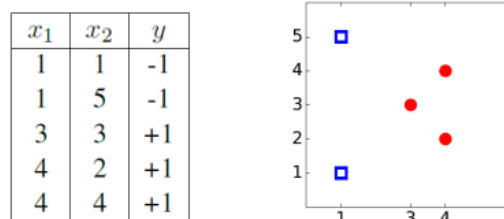


**Nunzio Emanuele Sgroi - [REDACTED]**

*Figure 24 - SVM vs. Perceptron Misclassification Rates Comparison*

## Task 2

**Q:** Consider we are learning a linear SVM for a training dataset shown in a table below, described with two features  $x_1$  and  $x_2$ , and binary target  $y$ .



*Figure 25 - Week 7 Task 2: Training Data for Linear SVM*

The classifier takes the form  $f(x; w_1; w_2; b) = \text{sign}(w_1x_1 + w_2x_2 + b)$

- Draw the decision boundary of the trained SVM, and identify the support vectors.
- Calculate the parameters of the trained SVM (i.e.,  $w_1$ ,  $w_2$ ,  $b$ ).

The decision boundary is a line that separates the classes, typically placed midway between the nearest opposing points on the  $x_1$ -axis. Points closest to this boundary are known as support vectors.

For this task, I developed a Python script that plots the decision boundary using the data from the table above. Additionally, this script provides the parameters of the Support Vector Machine (SVM) and identifies both the positive and negative support vectors.

```

## Nunzio Emanuele Sgroi - ██████████

# Data points and model fitting
X = np.array([[1, 1], [1, 5], [3, 3], [4, 2], [4, 4]])
y = np.array([-1, -1, 1, 1, 1])
clf = svm.SVC(kernel='linear', C=1000).fit(X, y)

# Plotting data points, decision boundary, and support vectors
plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)
plt.suptitle("Nunzio Emanuele Sgroi - ██████████")
plt.title("Decision Boundary")
ax = plt.gca()
xlim, ylim = ax.get_xlim(), ax.get_ylim()
xx, yy = np.linspace(xlim[0], xlim[1], 30), np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])
ax.scatter(clf.support_vectors[:, 0], clf.support_vectors[:, 1], s=100, linewidth=1, facecolors='none', edgecolors='k')
plt.show()

# Identify and print support vectors
support_vectors = clf.support_vectors_
dual_coefs = clf.dual_coef_[0]
positive_sv = support_vectors[(dual_coefs > 0) & (y[clf.support_] == 1)]
negative_sv = support_vectors[(dual_coefs < 0) & (y[clf.support_] == -1)]
print("-"*5)
print("Positive Support Vectors:", ' '.join(['{:.0f},{:.0f}'.format(sv[0], sv[1]) for sv in positive_sv]))
print("Negative Support Vectors:", ' '.join(['{:.0f},{:.0f}'.format(sv[0], sv[1]) for sv in negative_sv]))

# SVM parameters
print("w1 =", clf.coef_[0][0], "w2 =", clf.coef_[0][1], "b =", clf.intercept_[0])

```

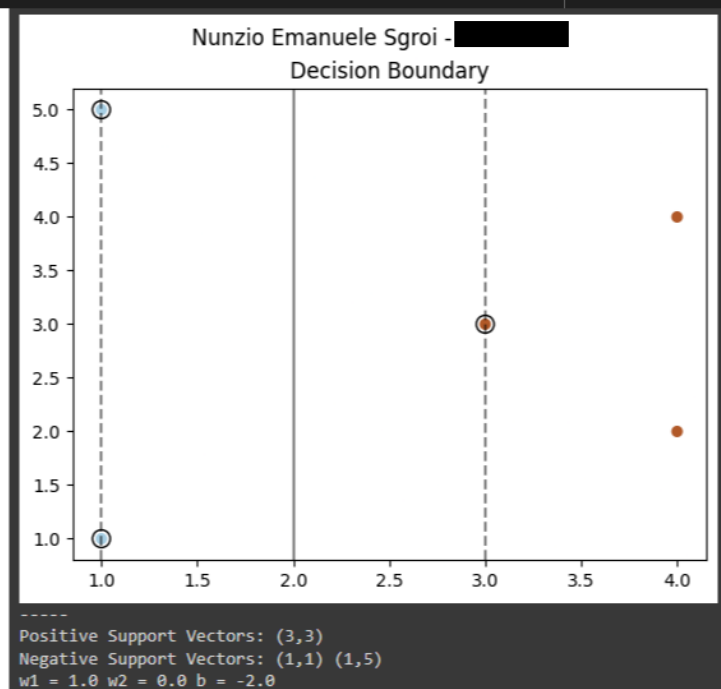


Figure 26 – Python Script for SVM Decision Boundary

To understand how this work, I have included a detailed explanation of the underlying processes. The analysis begins by identifying the support vectors: the data points at **(1,1)** and **(1,5)** are the negative support vectors, and the point at **(3,3)** is the positive support vector.

After identifying the support vectors, it is possible to calculate the parameters of the trained SVM (i.e.,  $w_1$ ,  $w_2$ ,  $b$ ). This is done by setting up simple equations based on these support vectors.

Student: Nunzio Emanuele Sgroi  
Student ID:

- For negative support vectors (where  $y = -1$ ):  $w_1x_1 + w_2x_2 + b = -1$
- For the positive support vector (where  $y = +1$ ):  $w_1x_1 + w_2x_2 + b = +1$

Applying these formulas for the support vectors identified, the next step would look as follow:

- From the point (1,1):  $w_1 \times 1 + w_2 \times 1 + b = -1$  (Equation 1)
- From the point (1,5):  $w_1 \times 1 + w_2 \times 5 + b = -1$  (Equation 2)
- From the point (3,3):  $w_1 \times 3 + w_2 \times 3 + b = +1$  (Equation 2)

The next step is to subtract the equation 1 from equation 2. This eliminates  $w_1$ , simplifying the equation to focus on  $w_2$  and  $b$ .

$$(w_1 + 5w_2 + b) - (w_1 + w_2 + b) = 0$$

$$5w_2 - w_2 = 0$$

$$4w_2 = 0; \text{ Therefore, } w_2 = 0$$

At this point, the equation  $w_2 = 0$ , can be substituted back into equation 1, giving a new equation.

$$w_1 + b = -1 \text{ (Equation 4)}$$

Now the equation 3 can be used to solve for  $w_1$  and  $b$ . With  $w_2 = 0$ , the equation becomes as follow:

$$3w_1 + b = 1 \text{ (Equation 5)}$$

It is now possible to solve for  $w_1$ , subtracting equation 4 from equation 5:

$$3w_1 - w_1 = 2$$

$$2w_1 = 2$$

$$w_1 = 1$$

Finally,  $w_1 = 1$  is substituted into equation 4 to find  $b$ :

$$1 + b = -1$$

$$b = -2$$

To summarise, the parameters for the trained SVM are:

- $w_1 = 1$
- $w_2 = 0$
- $b = -2$

## Week 8

---

### Task 1

**Q:** *Explain how a Multi-layer Perceptron can address the limitation of a single-layer Perceptron.*

A Multi-layer Perceptron (MLP) addresses the limitations of a single-layer Perceptron in several ways:

- **Learning Non-linear Functions:** Unlike a single-layer Perceptron that can only represent linear functions due to its single input and output layer, an MLP includes one or more hidden layers. This architecture equips MLPs to model complex, non-linear relationships, thereby enabling them to solve problems beyond linear separability (Bento, 2021).
- **Multiple-layers and Backpropagation:** The MLP's architecture, which consists of multiple layers of nodes with non-linear activation functions, paired with the backpropagation algorithm, allows for a refined adjustment of connection weights. This is a significant enhancement over the single-layer Perceptron's simpler weight adjustment mechanism, leading to improved learning from data (Bento, 2021).
- **Handling Large Amounts of Data and Complexity:** An MLP's capacity to handle vast and intricate data sets is a marked improvement over the single-layer Perceptron. This capability positions MLPs as a better choice for complex tasks such as image and speech recognition that are characteristic of modern data-rich environments (deepgram.com, 2023).
- **Efficiency in Prediction:** Once trained, MLPs can make predictions rapidly, which is crucial for real-time applications. They maintain high accuracy even with smaller data samples, demonstrating greater efficiency and versatility in various operational scenarios (Bento, 2021).

In conclusion, MLPs significantly overcome the limitations of single-layer Perceptrons by incorporating hidden layers for non-linear problem-solving, utilizing backpropagation for refined learning, managing larger and more complex data sets, and delivering efficient and accurate predictions in diverse applications.

### Task 2

**Q:** *The script applies Multi-layer Perceptron Classifier (Neural Network) to the Iris dataset. Search for the parameter `hidden_layer_sizes` in the code. You can use this parameter to define the number of neurons and layers in the network architecture:*

*$i^{\text{th}}$  element represents the number of neurons in the  $i^{\text{th}}$  hidden layer.*

*For example:*

- *`hidden_layer_sizes=(2,4)` defines a network with 2 hidden layers; first layer with 2 neurons and second layer with 4 neurons*
- *`hidden_layer_sizes=(20,4,2)` defines a network with 3 hidden layers; first layer with 20 neurons, second layer with 4 neurons, and third layer with 2 neurons*

*Try a few different configurations for the network architecture and discuss your observations in the logbook.*

*How does the number of neurons and hidden layers affect the performance of the network?*

*Can you find any architecture that yields 100% accuracy on test set?*

Student: Nunzio Emanuele Sgroi

Student ID:

The investigation into the Multi-layer Perceptron (MLP) classifier's performance on the Iris dataset with different network designs showed interesting results. Initially, the network was configured with a single hidden layer of 4 to 10 neurons. The best accuracy was found with **5** neurons, reaching **97.37%**. Increasing the number of neurons beyond this did not improve accuracy, indicating that a single layer with 5 neurons was enough to understand the dataset's patterns.

Further tests with two hidden layers retained the first layer at 5 neurons and varied the second layer's neuron count. A decline in accuracy was noted with just 2 or 3 neurons in the second layer; however, the accuracy rebounded to **97.37%** when the second layer contained **4** or more neurons. This indicates a minimum level of complexity needed in multi-layer networks to perform well.

A three-layer network was then tested, starting with the previously identified optimal two-layer setup. Adding a third layer and varying the number of neurons led to a notable finding: a configuration of **(5, 4, 5)** neurons achieved **100% accuracy** on the test set. Cross-validation, however, showed some variability in performance, averaging **0.98**, with a dip to **93.33%**, suggesting the model is strong but not flawless.

In conclusion, the MLP's effectiveness is greatly influenced by the number and size of layers and neurons. A balanced approach, such as the (5, 4, 5) structure, proved to be excellent for the test set, but cross-validation highlighted the importance of building a model that is accurate yet flexible enough for unseen data.

## Week 9

### Task 1

**Q:** Describe one method for setting Hyperparameters in k-NN classification method.

In k-NN classification, one method of setting hyperparameters is **Cross-Validation**. This technique involves dividing the dataset into several smaller groups, or “folds”. Each fold is used in turn as a validation set, while the rest serve as the training set. The k-NN model is trained and tested multiple times, with each fold getting a turn to be the validation set. The performance across these folds is then averaged, giving a reliable measure of how well the model might perform on new, unseen data (Allibhai, 2018).

Adjusting hyperparameters, like neighbour count in k-NN, improves model performance on validation sets. The goal is to optimize these settings for the best performance. After fine-tuning, the model is tested just once on a separate set to estimate its real-world effectiveness. While cross-validation benefits small datasets, it's resource-intensive for larger ones, striking a balance between detailed testing and resource use, ensuring the chosen settings enhance model performance and dependability.

### Task 2

**Q:** Based on the cross-validation results in the Introduction exercises above, choose the best value for k in k-NN. Justify your choice. Then retrain the image classifier using half the CIFAR10 training data, and test it on all the test data. What is the level accuracy?

The optimal value of k for the k-NN classifier, based on the cross-validation results, is **11**. This decision is backed by the highest average accuracy of **27.74%** achieved with **k=11**.

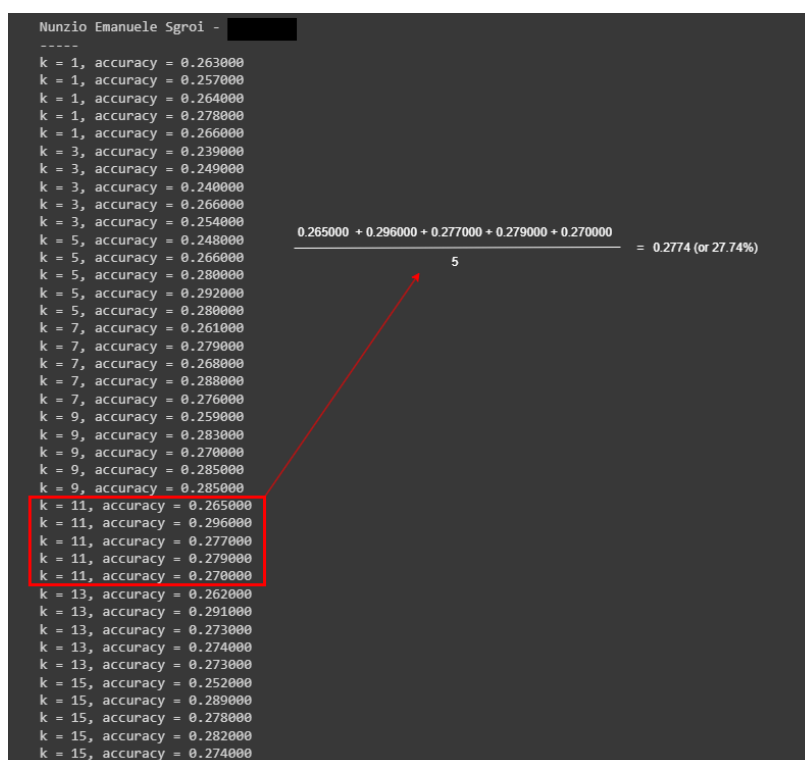


Figure 27- Cross-Validation Accuracy Scores for Selecting Optimal k in k-NN

Student: Nunzio Emanuele Sgroi

Student ID:

By using only half of the CIFAR10 training dataset and selecting the best k-value of **11** (identified through cross-validation), a **32.4%** accuracy (**0.324000**) was achieved with the image classifier. This was accomplished by initially reloading the dataset, followed by reducing the training set to **25,000** images, reshaping these images into the appropriate format, and then employing the k-NN classifier with k set to **11**. When this classifier was tested on the complete test set of **10,000** images, it correctly predicted **3,240** cases, resulting in the mentioned accuracy rate.

```
## Nunzio Emanuele Sgroi - XXXXXXXXXX

best_k= 11 #obtained from the cross-validation
X_train, y_train, X_test, y_test = load_CIFAR10('/content/gdrive/My Drive/Colab Notebooks/CIFAR10/') # Load data

# Set training value
num_training = 25000
mask = list(range(num_training))
X_train = X_train[mask]
y_train = y_train[mask]

# Set testing value
num_test = 10000
mask = list(range(num_test))
X_test = X_test[mask]
y_test = y_test[mask]

# Reshape data
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))

# Train and predict classifier
classifier = KNearestNeighbor()
classifier.train(X_train, y_train)
y_test_pred = classifier.predict(X_test, k=best_k)

# Calculate and display results
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / num_test
print('Got %d / %d correct => accuracy: %f' % (num_correct, num_test, accuracy))
```

```
/content/gdrive/My Drive/Colab Notebooks/CIFAR10/data_batch_1
/content/gdrive/My Drive/Colab Notebooks/CIFAR10/data_batch_2
/content/gdrive/My Drive/Colab Notebooks/CIFAR10/data_batch_3
/content/gdrive/My Drive/Colab Notebooks/CIFAR10/data_batch_4
/content/gdrive/My Drive/Colab Notebooks/CIFAR10/data_batch_5
/content/gdrive/My Drive/Colab Notebooks/CIFAR10/test_batch
Got 3240 / 10000 correct => accuracy: 0.324000
```

Figure 28 - Image Classifier tested on half CIFAR10 training data

## Week 10

### Task 1

**Q:** For the convolution operation given below, compute the destination pixel value in the output image by applying the 3x3 filter to the source (input) image. Provide the details of your calculations.

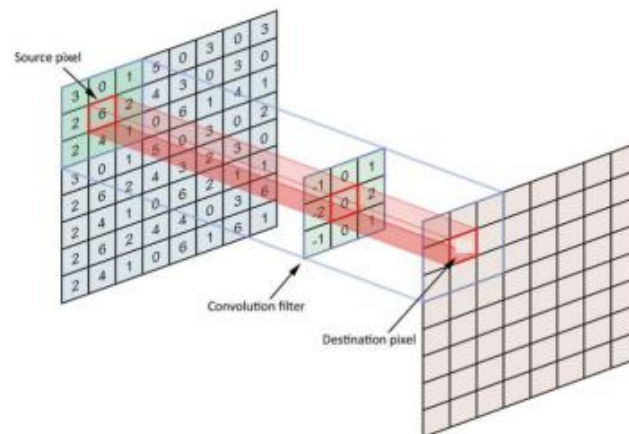


Figure 29 - Week 10, Task 1: Convolution Filter Operation

Given a source image and a convolution filter, the goal is to compute the value of a destination pixel in the output feature map. The calculation involves few steps, starting with aligning the filter with the image region (as highlighted in the image above), multiply each element of the filter by the corresponding element in the image region and sum all the products.

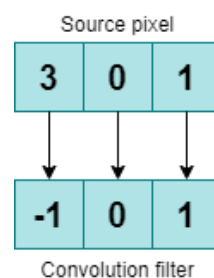


Figure 30 - 3x3 Convolution Filter Example for Calculation

For example, the number 3 from the top-left cell of the source pixel region corresponds with the number -1 from the top-left cell of the convolution filter. These two numbers are then multiplied together. The full calculation is detailed below:

$$\begin{aligned}
 &(-1 \times 3) + (0 \times 0) + (1 \times 1) + (-2 \times 2) + (0 \times 6) + (2 \times 2) + (-1 \times 2) + (0 \times 4) + (1 \times 1) = \\
 &= -3 + 0 + 1 - 4 + 0 + 4 - 2 + 0 + 1 = \\
 &= \mathbf{-3}
 \end{aligned}$$

Figure 31 - Convolution Operation Calculation

The convolution operation results in a destination pixel value of **-3**, which will be positioned centrally in the output feature map, aligning with the filter's location on the source image.

Student: Nunzio Emanuele Sgroi  
Student ID:

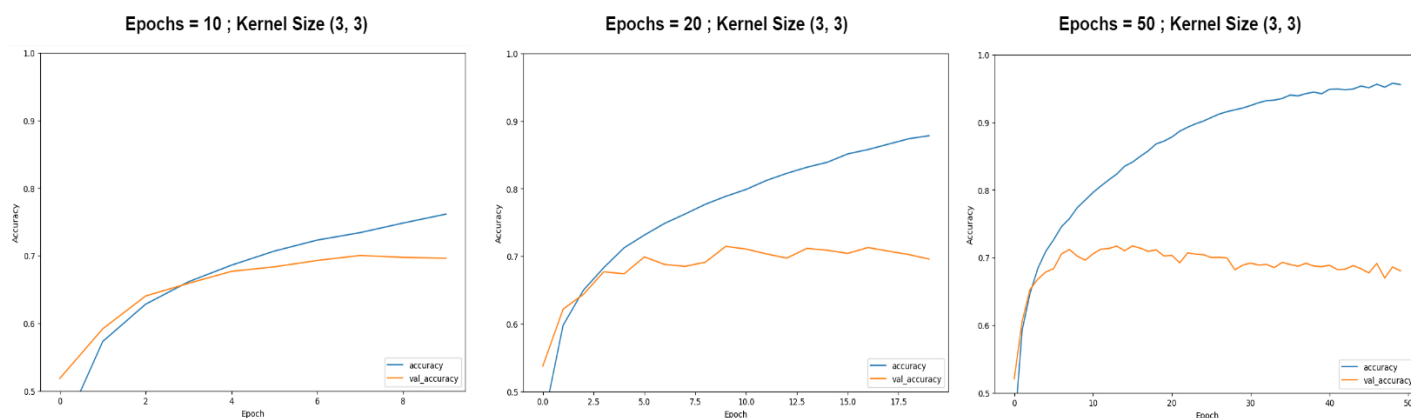
## Task 2

**Q:** Questions from the source code provided:

- Train the CNN model for a few more iterations (you can do this by changing the number of epochs in `model.fit`, then click on Runtime and Run all)
- Try it for 10, 20, 50 epochs.
- How does the accuracy change? Discuss your observations.
- If accuracy is improving, explain why. If accuracy is getting worse, why do you think that is?
- Change the kernel size from (3, 3) to (5, 5) in your CNN model (for all layers.Conv2D layers)
- Would that help to improve the accuracy?

After training the CNN model with 10, 20 and 50 epochs and a kernel size of (3, 3), it was observed that as the number of epochs increases, the model's training accuracy generally improves. This trend is indicative of the model's growing proficiency in learning from the training data. From 10 to 20 epochs, the model shows improved performance on both training and validation datasets, suggesting that with more iterations, the model learns to generalize better.

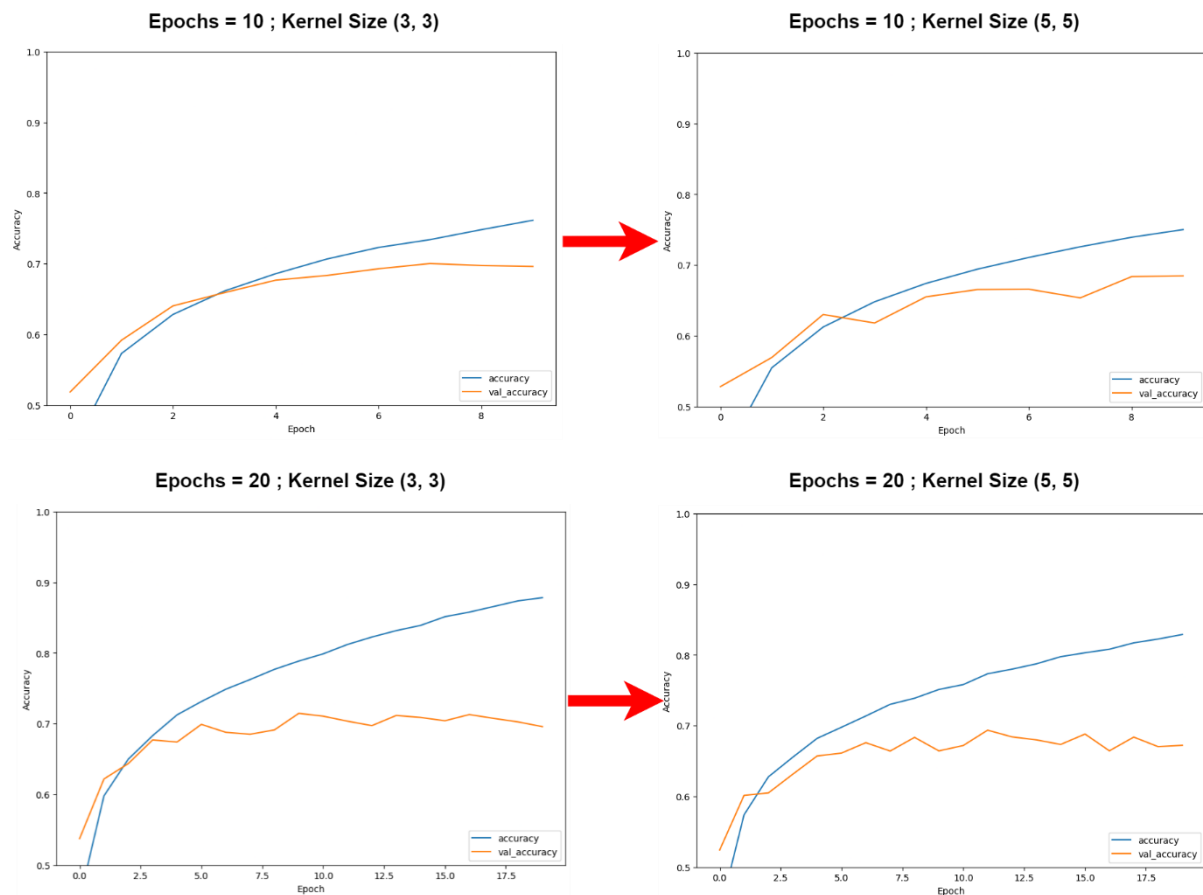
However, upon extending the training to 50 epochs, it was observed a divergence between training and validation accuracy. The training accuracy continued to rise, indicating that the model was increasingly successful at recognizing the data it was trained on. However, the improvement in accuracy on the validation set, which consists of data not seen by the model during training, began to stall and then showed a slight decline, compared with the previous tests. This suggests that the model's ability to generalize to new data is not keeping pace with its ability to fit the training data, and this is called overfitting.



Nunzio Emanuele Sgroi - [REDACTED]

Figure 32 - Training Accuracy vs. Validation Accuracy Across Epochs

Further experiments with a kernel size of (5, 5) for 10 and 20 epochs revealed a marginal decrease in accuracy when compared to the smaller (3, 3) kernel. This could be due to the larger kernel size capturing more global features of the image, which might not be as essential for the CIFAR10 dataset, where smaller and more detailed features are significant for classification. A larger kernel may also require more data to effectively learn and generalize these features, which was not provided in the current training regime. The 20-epoch training with the larger kernel did not improve validation accuracy, confirming that for this specific dataset and model architecture, a larger kernel size does not contribute positively to model generalization.



Nunzio Emanuele Sgroi - [REDACTED]

Figure 33 - Epoch-to-Accuracy Comparisons Before and After Adjusting CNN Parameters

In conclusion, the task revealed that a smaller kernel size of (3, 3) was more suitable for this particular dataset. Additionally, it was evident that training beyond a certain number of epochs can lead to overfitting, where the model memorizes the training data and its performance on new, unseen data deteriorates.

## Week 11

### Task 1

**Q:** A teacher is interested to work out how time spent writing a logbook affects the grades. He develops a regression model to predict what score someone will get from a logbook based on how long they spent on it. He has the data shown on the opposite:

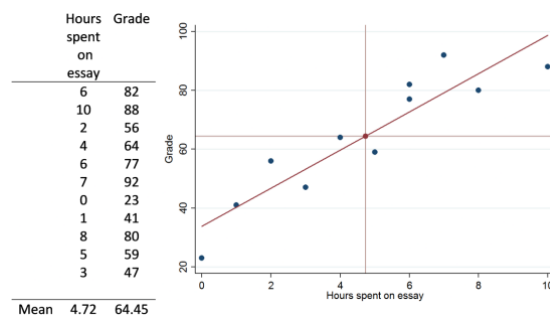


Figure 34 - Week11, Task 1: Regression Model Data

The best regression model he finds is:  $y = 6.49x + 30.18$ :

Calculate the Least Squares Error for this model.

The Least Squares Error is calculated as the sum of the squares of the differences between the observed values (grades) and the predicted values (grades predicted by the regression model). The formula for LSE is:

$$\sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Figure 35 - Formula to calculate LSE

This involves summing the squares of the differences between each grade ( $y_i$ ) and the grade predicted by the regression model ( $\hat{y}_i$ ) for the corresponding hours spent on the essay.

Giving the regression model  $y = 6.49x + 30.18$ , the calculation for the Least Squares Error based on the first row of the table, where 6 hours were spent on the essay and a grade of 82 was received, proceeds as follows:

#### Calculation of the predicted grade

The predicted grade when 6 hours are spent on the essay is calculated as follows:

$$\hat{y} = 6.49 \times 6 + 30.18 = 68.94$$

#### Calculation of the error

The error is the difference between the actual grade and the predicted grade:

$$\text{Error} = y_i - \hat{y}_i = 82 - 68.94 = 13.06$$

#### Squared error

Squaring the error gives the squared error for this observation:

$$\text{Squared Error} = 13.06^2 = 170.56$$

Figure 36 - Example Calculation of LSE

After repeating this process for all the data points, all the squared errors calculated needs to be summed and the result will be the Least Squares Error.

I created a Python program to do the calculations faster and more accurately, which helped get the precise Least Squares Error.

```

## Nunzio Emanuele Sgroi - ██████████

# Dataset: hours spent and grades
hours_spent = np.array([6, 10, 2, 4, 6, 7, 0, 1, 8, 5, 3])
grades = np.array([82, 88, 56, 64, 77, 92, 23, 41, 80, 59, 47])

# Regression model function
def predict_grade(hours, slope=6.49, intercept=30.18):
    return slope * hours + intercept

# Calculation predicted grades
predicted_grades = predict_grade(hours_spent)

# Calculation errors
errors = grades - predicted_grades

# Calculation squared errors
squared_errors = errors**2

# Calculation LSE
lse = np.sum(squared_errors)

print(f"The Least Squares Error is: {lse:.2f}")

The Least Squares Error is: 868.30

```

Figure 37 - Python code that compute LSE

Based on the calculation explained above, the Least Squares Error for this model is: **868.30**.

## Task 2

**Q:** Table below shows some data from the early days of the Italian clothing company Benetton. Each row in the table shows Benetton's sales for a year and the amount spent on advertising that year. In this case, our outcome of interest is sales—it is what we want to predict. If we use advertising as the predictor variable, we can find a linear regression model which would give estimates of the sales.

Which one of the following will the best regression model? Justify your answer.

- $Sales = 22.60 \times Advertising + 169.2$
- $Sales = 23.42 \times Advertising + 167.7$
- $Sales = 23.10 \times Advertising + 168.1$

Year	Sales (Million Euro)	Advertising (Million Euro)
1	651	23
2	762	26
3	856	30
4	1,063	34
5	1,190	43
6	1,298	48
7	1,421	52
8	1,440	57
9	1,518	58

Figure 38 - Week 11, Task 2: Regression Model Data

Student: Nunzio Emanuele Sgroi  
Student ID:

To identify the optimal regression model from the presented choices, the same methodology as in the previous task is applied: the Least Squares Error is computed for each model using the standard formula.

The model that would produce the lowest LSE, is the best regression model because it indicates the smallest difference between the observed sales and the predicted sales. I developed a python code that would calculate the LSE for each model, returning the model with the lowest outcome as the best model:

```
## Nunzio Emanuele Sgroi - [REDACTED]

# Dataset
advertising_spend = np.array([23, 26, 30, 34, 43, 48, 52, 57, 58])
sales = np.array([651, 762, 856, 1063, 1190, 1298, 1421, 1440, 1518])
smallest_lse = float('inf')
best_model = 0

# Regression models
models = [
    {"slope": 22.60, "intercept": 169.2},
    {"slope": 23.42, "intercept": 167.7},
    {"slope": 23.10, "intercept": 168.1}
]

# Function to calculate LSE
def calculate_lse(sales, advertising_spend, slope, intercept):
    predicted_sales = slope * advertising_spend + intercept
    squared_errors = (sales - predicted_sales) ** 2
    return np.sum(squared_errors)

# Calculating and printing the LSE for each model
for i, model in enumerate(models, start=1):
    lse = calculate_lse(sales, advertising_spend, model['slope'], model['intercept'])
    print(f"Model {i}: LSE = {lse:.2f}")
    if lse < smallest_lse:
        smallest_lse = lse
        best_model = i

# Printing best regression model
print("-"*5)
print(f"Model {best_model} is the best regression model")
print(f"Equivalent to: Sales = {models[best_model - 1]['slope']} x Advertising + {models[best_model - 1]['intercept']}")

Model 1: LSE = 29224.96
Model 2: LSE = 18804.03
Model 3: LSE = 20448.82
-----
Model 2 is the best regression model
Equivalent to: Sales = 23.42 x Advertising + 167.7
```

Figure 39 - Python Code that compute LSE and Identifies Optimal Model with lowest LSE

Running the python script shown above, it is observed that the LSE for each regression model is:

- Sales = 22.60 × Advertising + 169.2 = **29224.96**
- Sales = 23.42 × Advertising + 167.7 = **18804.03**
- Sales = 23.10 × Advertising + 168.1 = **20448.82**

Therefore, as highlighted above, the best regression model is the second one in the list. This model has the smallest LSE (**18804.03**), indicating that its sales predictions are most closely aligned with the actual sales figures.

Student: Nunzio Emanuele Sgroi  
Student ID:

## Task 3

**Q:** For this task, download *Regression\_Advertising.csv* from Blackboard. This is a simple dataset that contains, in unit of £, the marketing spend along with the company sales for each month for a Software company. Using the advertising data, model the linear relationship between the marketing budget and sales. Then answer the following questions:

- How much sale can we expect when the marketing budget is zero?
- For a marketing spend of £7,500, how much sale can we expect?

The linear regression model for the given dataset, formulated the following results:

- Expected sales when marketing budget is zero: **£1383.47**
- Predicted sales for a marketing spend of £7500: **£81049.94**

Mathematically, the model is based on the equation of a straight-line  $y = mx + b$ :

- $y$  is the dependent variable — in this case, the sales.
- $x$  is the independent variable — the marketing spend.
- $m$  is the slope of the line — it represents the change in sales for each unit change in marketing spend.
- $b$  is the y-intercept — it predicts the sales when the marketing spend is zero.

The model calculates the optimal values for  $m$  and  $b$  by minimizing the residual sum of squares between the actual and predicted sales in the training data. This process is known as fitting the model.

To estimate sales when the marketing budget is zero, the intercept  $b$  of the linear equation provides the necessary information. While, to predict the sales for a marketing spend of £7,500, the value of  $x$  is set to 7,500 in the model's equation. The corresponding sales  $y$  are calculated as follows:

- $y = (m \times 7500) + b$

I modified the code provided for training and testing purposes. Additionally, it now creates a scatter plot with a regression line, showing the position of the calculated results on the graph, along with the dataset points.

```
## Nunzio Emanuele Sgroi - [REDACTED]
# Predict sales when the marketing budget is zero
print(f'Expected sales when marketing budget is zero: {linear_regressor.intercept_[0]:.2f}')
# Predict sales for a marketing spend of £7,500
predicted_sales_for_7500 = linear_regressor.predict([[7500]])[0][0]
print(f'Predicted sales for a marketing spend of £7500: {predicted_sales_for_7500:.2f}')

Expected sales when marketing budget is zero: 1383.47
Predicted sales for a marketing spend of £7500: 81049.94
```

Figure 40 - Python Code with results for Week 11, Task 3

```

## Nunzio Emanuele Sgroi - ██████████

# Plot original training set results
plt.scatter(marketing_spend, sales, color = 'red')
plt.plot(marketing_spend, predicted_sales, color = 'blue')
plt.title('Sales vs. Marketing Spend')
plt.xlabel('Marketing Spend (£)')
plt.ylabel('Sales (£)')
plt.suptitle('Nunzio Emanuele Sgroi - ██████████')

# Show expected sales when marketing budget is zero
plt.annotate(f'£{linear_regressor.intercept_[0]:.2f}',
            xy=(0, linear_regressor.intercept_[0]),
            xytext=(marketing_spend.max()/10, linear_regressor.intercept_[0] + (sales.max()/10)),
            arrowprops=dict(facecolor='green', shrink=0.05),
            ha='center')

# Show predicted sales for a £7,500 marketing spend
plt.annotate(f'£{predicted_sales_for_7500:.2f}',
            xy=(7500, predicted_sales_for_7500),
            xytext=(7500, predicted_sales_for_7500 + (sales.max()/20)),
            arrowprops=dict(facecolor='green', shrink=0.05),
            ha='center')

# Set plot limits: necessary to show expected sales when marketing budget is zero
plt.xlim(left=0, right=marketing_spend.max() + 10000)
plt.ylim(bottom=0, top=sales.max() + 10000)

plt.show()

```

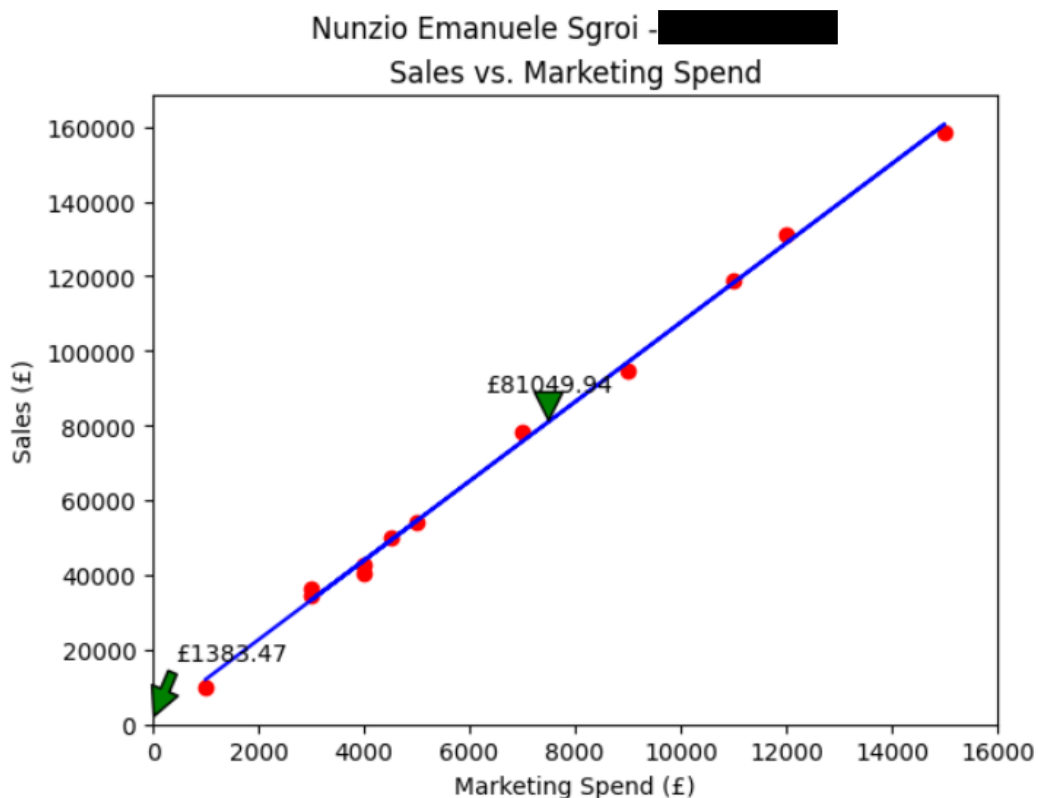


Figure 41 - Python Plot of Sales Regression Based on Marketing Spend

## Task 4

**Q:** Modify the Python script to compute the cost function  $J$  for the flitted line in Task 2. Provide a snapshot of your additions to the script. Also provide the results.

To compute the cost function  $J$  for the given linear regression models in task 2, I used the following formula:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Figure 42 - Formula to calculate Cost Function  $J$

In this formula,  $h_{\theta}(x^{(i)})$  represents the prediction from the linear model for each data point,  $y^{(i)}$  is the actual value, and  $m$  is the total number of data points.

I implemented a python function, `calculate_cost`, to execute the necessary computational steps. These steps included calculating the hypothesis for each data point, determining the error for each data point, squaring these errors, summing the squared errors, and ultimately computing the cost function.

The computed results for each model are as follows:

- **Model 1** (Sales = 22.60 × Advertising + 169.2): Cost function  $J(\Theta_0, \Theta_1) = 1623.61$
- **Model 2** (Sales = 23.42 × Advertising + 167.7): Cost function  $J(\Theta_0, \Theta_1) = 1044.67$
- **Model 3** (Sales = 23.10 × Advertising + 168.1): Cost function  $J(\Theta_0, \Theta_1) = 1136.05$

The outcomes revealed Model 2 as the most suitable option, aligning with conclusions stated in Task 2., because it registered the minimum values for both the cost function  $J$  and the Least Squares Error (LSE). Therefore, the cost function  $J$  for the flitted line in task 2 is **1044.67**.

```

## Nunzio Emanuele Sgroi - ██████████

# Dataset from task 2
advertising = np.array([23, 26, 30, 34, 43, 48, 52, 57, 58])
sales = np.array([651, 762, 856, 1063, 1190, 1298, 1421, 1440, 1518])
smallest_j = float('inf')
best_model = 0
models = [
    {"slope": 22.60, "intercept": 169.2},
    {"slope": 23.42, "intercept": 167.7},
    {"slope": 23.10, "intercept": 168.1}
]

# Function to calculate the cost function J
def calculate_cost(slope, intercept, x_values, y_values):
    m = len(x_values)
    predictions = intercept + slope * x_values
    squared_errors = (predictions - y_values) ** 2
    cost = (1 / (2 * m)) * np.sum(squared_errors)
    return cost

# Loop to calculate cost function J for each model
for i, model in enumerate(models, start=1):
    cost = calculate_cost(model['slope'], model['intercept'], advertising, sales)
    if cost < smallest_j:
        smallest_j = cost
        best_model = i

# Display results
for model, cost in detailed_costs.items():
    print(f"{model}: Cost function J(Theta_0, Theta_1) = {cost:.2f}")

print("-"*5)
print(f"Model {best_model} is the best regression model")
print(f"Equivalent to: Sales = {models[best_model - 1]['slope']} x Advertising + {models[best_model - 1]['intercept']}")

Model 1: Cost function J(Theta_0, Theta_1) = 1623.61
Model 2: Cost function J(Theta_0, Theta_1) = 1044.67
Model 3: Cost function J(Theta_0, Theta_1) = 1136.05
-----
Model 2 is the best regression model
Equivalent to: Sales = 23.42 x Advertising + 167.7

```

Figure 43 - Python Code that Compute Cost Function  $J$

Student: Nunzio Emanuele Sgroi  
Student ID:

## References

---

- Allibhai, J. (2018, September 26). *Building a k-Nearest-Neighbors (k-NN) Model with Scikit-learn*. Retrieved from towardsdatascience.com: <https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a>
- Bento, C. (2021, 09 21). *Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis*. Retrieved from towardsdatascience.com: <https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141>
- deepgram.com. (2023, 09 29). *Perceptron*. Retrieved from deepgram.com: <https://deepgram.com/ai-glossary/perceptron>
- Google. (2022). *k-Means Advantages and Disadvantages*. Retrieved from Google Developer: <https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages#:~:text=k%2Dmeans%20has%20trouble%20clustering,cluster%20instead%20of%20being%20ignored.>
- IBM. (no data). *K-Nearest Neighbors Algorithm*. Retrieved from ibm: <https://www.ibm.com/topics/knn#:~:text=Lower%20values%20of%20k%20can,with%20higher%20values%20of%20k.>
- Kumar, S. (2020). *Understanding K-Means, K-Means++ and, K-Medoids Clustering Algorithms*. Retrieved from Towards Data Science: <https://towardsdatascience.com/understanding-k-means-k-means-and-k-medoids-clustering-algorithms-ad9c9fbf47ca>
- Kundu, R. (2022, December 16). *F1 Score in Machine learning: Intro & Calculation*. Retrieved from v7labs: <https://www.v7labs.com/blog/f1-score-guide>
- Roy, B. (2020, April 6). *All about Feature Scaling*. Retrieved from All about Feature Scaling: <https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35>