EE 565 - Machine Learning I
Project 0 Report

Jorge A. Garcia

***Abstract*—Different algorithms are implemented to solve the essential tasks in machine learning. For the task of unsupervised learning, K-Means is used to find different clusters within a data set. Polynomial fitting and K-Nearest Neighbors are used for supervised regression and classification accordingly.**

***Index Terms*—Machine Learning, Data Science**

I. Introduction

The learning methods that can be taken in machine learning can be divided into two groups: supervised and unsupervised learning. Supervised learning consists of training models based off of known results, whereas unsupervised allows the model to converge to a result without any prior knowledge. Within supervised learning, the two principal tasks that are solved as those of classification and regression. Classification attempts to assign a label to a data point, whereas regression outputs a real-valued number dependant on the input data. The most common unsupervised task is that of clustering, in which the goal is to find possible groupings within a data set.

In this project, we implement three machine learning algorithms that are representative of each task. We use K-means to approach the problem of clustering data, using both batch and on-line learning methods. An application of this algorithm is then used to compress images to different bit rates. We the implement polynomial curve fitting, both regularized and unregularized, to approach a regression task, and K-Nearest Neighbors for the task of classification using three different data sets.

II. Problem 1: Batch K-means

A K-Means class is implemented in Python for the following set of problems. Batch learning is used to find the cluster centers. The data set used consists of two circular Gaussian distributions, with N points evenly distributed between the two distributions. One distribution is centered about the origin, with means $\mu_{x_1}^{(0)} = \mu_{x_2}^{(0)} = 0$, and the second having means $\mu_{x_1}^{(1)} = \mu_{x_2}^{(1)} = 5$. Both have a variance of $\sigma^2 = 3$.

A. Variation in number of data points

In order to describe the performance of K-means clustering, the error as a function of number of data points is measured. The number of data points N is varied in the range $[10, 500]$ with a step size of 2. K-means is used to find two clusters ($K = 2$) amongst the data. The error is then defined to be the sum of Euclidean distances between the centroids $\boldsymbol{\mu_k}$ found and the actual means of the circular Gaussian distribution

$$E = \sum_{k=1}^{K} \left|\left| \boldsymbol{\mu_k} - \boldsymbol{\mu_{\text{actual}}} \right|\right| \tag{1}$$

For each set of N data points, K-means is used 10 times as to have a statistical description of the behavior of the algorithm with the data. The average error for each N points is then calculated and plotted, as seen in Figure 1. It can be observed that, on average, the error in converging to the true cluster centers exponentially decreases as the number of data points increases.

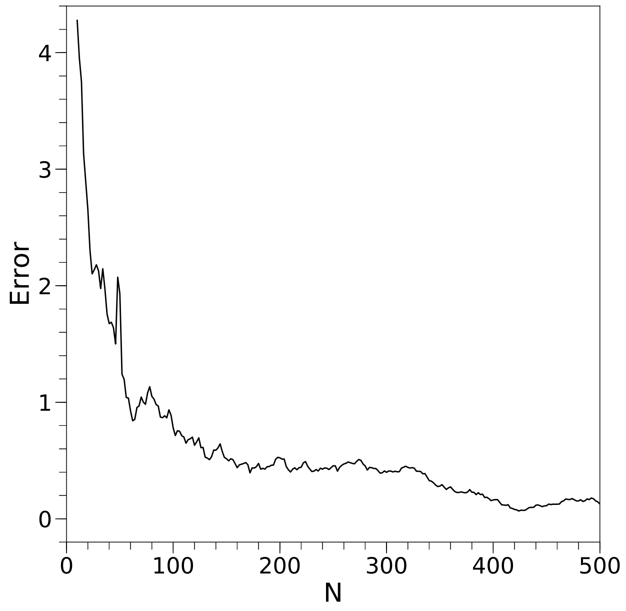


Fig. 1. Plot of the average error of K-means with $K = 2$ as a function of number of data points N of two circular Gaussian distributions.

B. Initialization of centers

Initialization of the centers when applying K-means can, depending on the data in question, affect the end-result. Different initial points can converge to different cluster centers. If a given initial center is especially bad, it can result with no data points being associated with it and when updating its location it will result in a division by 0. This would give a NaN value for the new location of the centroid, and will require its reinitilization or removal.

C. *Iterations for convergence*

For a data set of $N = 50$ points, 1000 trials of using K-means with $K = 2$ are done to have a statistical description of the number of iterations needed for the algorithm to converge. The algorithm was said to have converged when the average Euclidean distance between the updated and previous centers was less than a threshold ϵ

$$\frac{1}{K}\sum_{k=1}^{K}\left|\left|\boldsymbol{\mu}_{\boldsymbol{k}}^{(\mathbf{new})}-\boldsymbol{\mu}_{\boldsymbol{k}}^{(\mathbf{old})}\right|\right|<\epsilon \tag{2}$$

Which for this problem, $\epsilon = 1\times10^{-5}$. On average, the number of iterations needed to find the cluster centers was of 3.5, with the maximum being 7 iterations and the minimum being 2 iterations.

D. *Variation of K groups*

Evaluation of the cost function is dependant on the number of K clusters being searched for. For a data set of $N = 50$ points, K-means is used with a varying number of clusters in the range $[2, 20]$ in increasing steps of 1. The algorithm is fitted 100 times for each K value to obtain a statistical description of the cost for each number of clusters. The average cost of each K is then calculated, and plotted alongside its corresponding standard deviation in Figure 2. It can be observed that the cost exponentially decays as the number of centroids K increase. The variance in the cost for $K > 2$ is consistently larger than in the true case $K = 2$. This leads to believe that the variance in repeated trials of K-means is a better metric for determining the ideal number of clusters than that of cost alone.

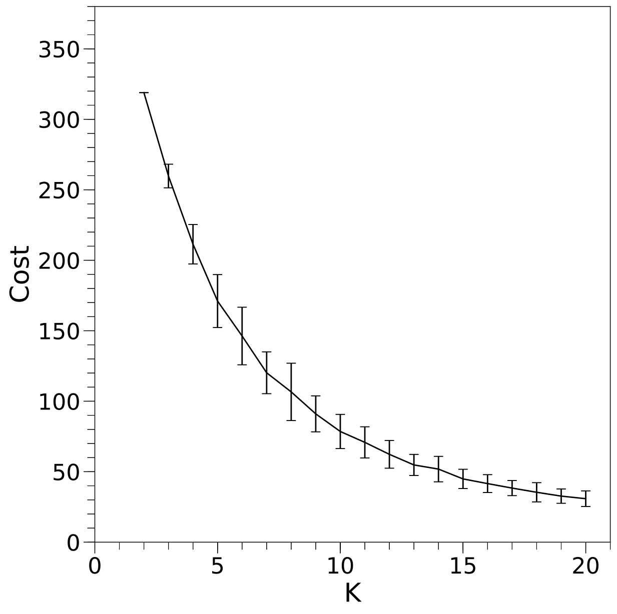


Fig. 2. Plot of the average cost of K-means as a function of number of clusters K on a data set of two circular Gaussian distributions.

E. *Clustering with different centers*

Using a data set of $N = 50$ points, three different subsets of 5 random data points are used to initialize the centers to be found for K-means with $K = 5$. Figure 3 shows the resulting clusters from each set of initial guesses. Each initial point is represented with an orange cross, and is linked to its converged location represented with a red cross. As seen in Figure 3, when using an inappropriate choice of K, different initial guesses converge to different points which results in completely different clusters.

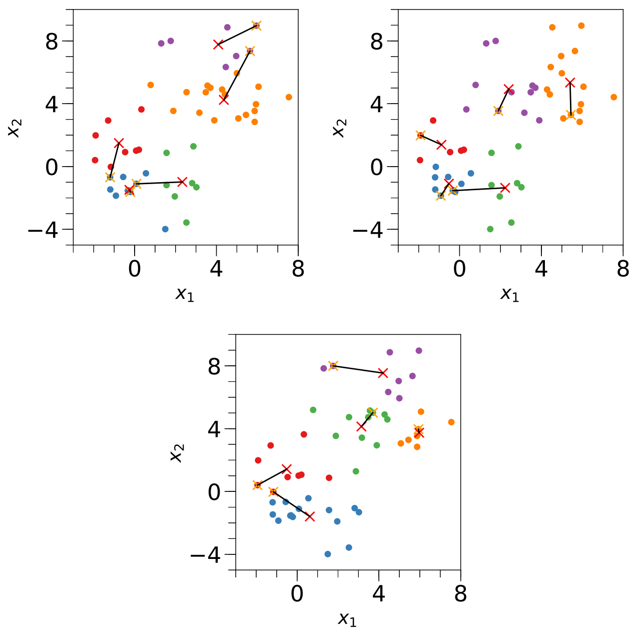


Fig. 3. Resulting clusters and centroids for three different initial guesses for K-means with $K = 5$. The orange cross represents the initial centroid, and the red cross its final location.

III. Problem 2: On-line K-means

The same K-Means class used before also contains an On-line learning routine, which is instead used in the following problem. The data set used is two circular Gaussian distributions, with $N = 500$ points distributed equally between them. Same means $\mu_{x_1}^{(0)} = \mu_{x_2}^{(0)} = 0$ and $\mu_{x_1}^{(1)} = \mu_{x_2}^{(1)} = 5$ are used, as is a variance of $\sigma^2 = 3$ for both.

A. *Learning rate with earliest convergence*

To find the best learning rate for this data set, the convergence of K-means is fixed to $\epsilon = 1\times10^{-1}$ and the same metric as in Equation 2 is used. A high convergence threshold was used due to the more volatile predictions obtained from on-line learning. The learning rate η is varied in log-space, with $\log\eta$ being in the range $[-3, -1]$ with an increasing step size of 0.1 per iteration. Learning at each learning rate is done 10 times in order to have a statistical description of the number of epochs needed to converge. A maximum number of 20 epochs is used to limit the time spent per learning rate. The resulting average number of epochs per log learning rate and its corresponding standard deviation can be seen in Figure 4. As seen in the plot, the learning rate that results in the lowest number of epochs needed to converge is $\eta = 1 \times 10^{-1.9} \approx 0.013$

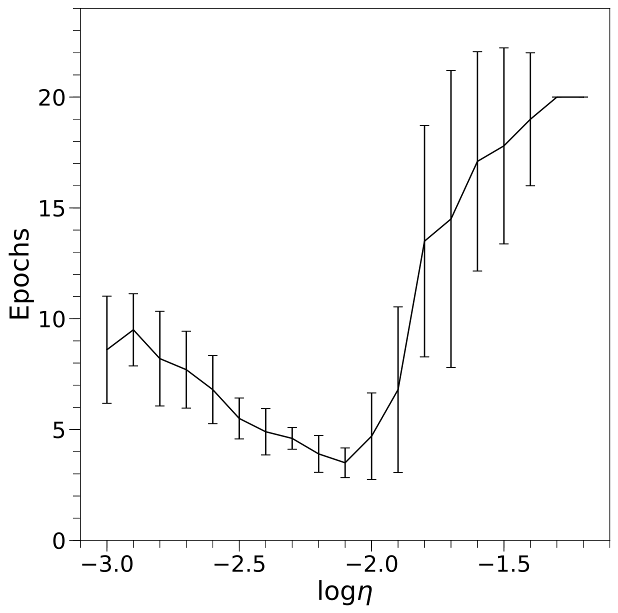


Fig. 4. Plot of the average number of epochs needed for convergence as a function of the log of the learning rate.

IV. Problem 3: K-means Application: Image Segmentation

For this section, the previously mentioned K-means class is used with batch learning. The data used consists of three different images, provided by the instructor.

We explore image compression/color reparameterization as an application of K-means. The pixels that conform a NxM image are unraveled into a $(NxM)x3$ matrix, which represents a data set consisting of all the pixels of the image in RGB space. K-means is then used to find $K = 2^b$ color centroids to remap the image, with b being the number of bits to compress the image to.

Due to the large amount of pixels being handled (with the smallest image containing > 250000 pixels), a low convergence threshold of $\epsilon = 1 \times 10^{-2}$. Lower maximum number of iterations are used, as the large number of data points results in better updates and, given the discussion in Problem 1a, lower errors. Before determining the most optimal bit rate, images were compressed between 2 and 7 bits to observe change between neighboring bit-rates. Scripts were then reduced to compute only the bit rates necessary as to save computing time.

A. Optimal bits for compression: "machine-learning-1.png"

The image compression routine is applied to the image "machine-learning-1.png" and remapped to 2 and 3 bits, shown in Figure 5. The majority of the image is very well represented due to the monotonous color palette present, with the exception being the "chip" on the side of the head. This is due to the complex details and various colors in this section, and they aren't very well captured at these bit rates. Also shown in Figure 5 is the image remapped to $K = 32$ colors, corresponding to 5 bits, which I considered gives a good enough representation. Particularly, the structure in the area of the chip is largely conserved and details can be better resolved.

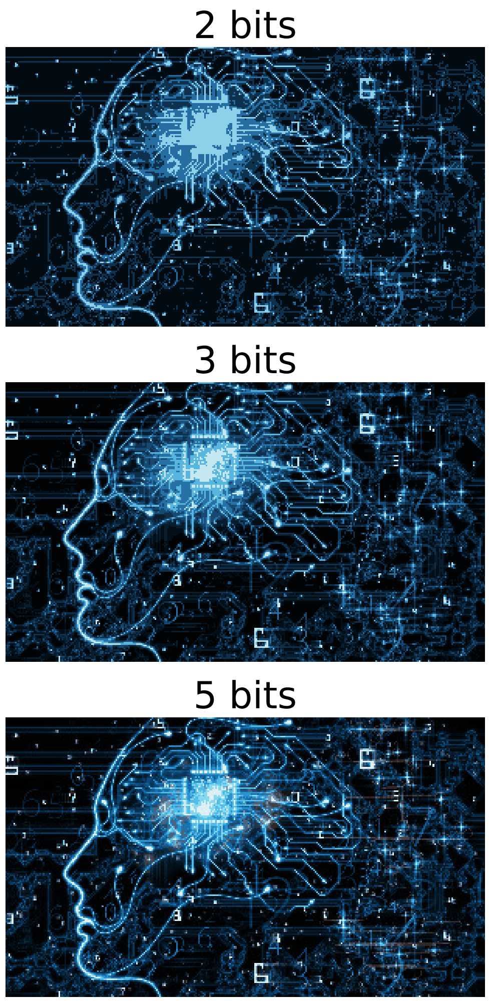


Fig. 5. The "machine-learning-1.png" image compressed to 2, 3 and 5 bits.

B. Optimal bits for compression: "Nature-Brain.png"

The image compression routine is applied to the image "Nature-Brain.png" and remapped to 2 and 3 bits, shown in Figure 6. The image is almost perfectly well represented, largely due to the color pattern of the original image being mostly composed of slightly different shades of gray. While detail may not be lacking, where the compression fall's short is in capturing the few shades of brown present in some areas. Also plotted in Figure 6 is the image remapped to $K = 64$ colors, corresponding to 6 bits, which I considered gives a good enough representation. This bit-rate allowed the brown-tones to be conserved, giving a closer appearance to the original image.

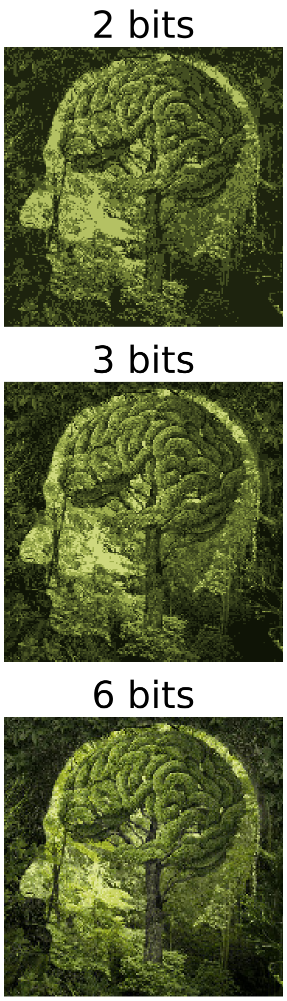


Fig. 6. The “Nature-Brain.png” image compressed to 2, 3 and 6 bits.

C. Optimal bits for compression: “nature-1.png”

The image compression routine is applied to the image “nature-1.png” and remapped to 2 and 3 bits, shown in Figure 7. This is a more complex image than the previous two, due to the amount of details present and variety of colors. They do not represent the image very well; the 2-bit image fails to capture any greens, whereas the 3-bit lacks enough detail. Also plotted in Figure 7 is the image remapped to $K = 128$ colors, corresponding to 7 bits, which I considered a good enough representation. While it only cuts the original number of colors in half, it effectively captures the necessary colors and details.

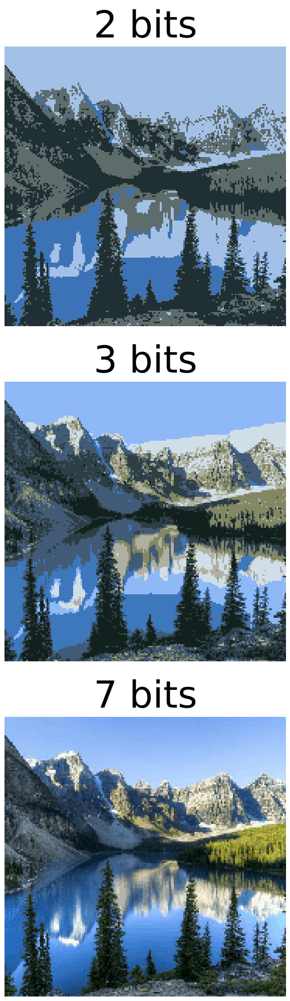


Fig. 7. The “nature-1.png” image compressed to 2, 3 and 7 bits.

D. Transferring compression centroids: From “nature-1.png” to others

The image compression routine is applied to the image “nature-1.png”, and its center colors are used to remap the images “machine-learning-1.png” and “Nature-Brain.png” to 2 and 3 bits, shown in Figures 8 and 9 respectively.

For “machine-learning-1.png”, the 2-bit representation has a washed-out appearance that is too dissimilar to the original image, but the 3-bit version does fairly well in color and detail. Also plotted in Figure 8 is the image remapped to $K = 32$ colors, corresponding to 5 bits, which I considered a good enough representation. While it is impossible to obtain the original color scheme, the essence of it is there and the details in the area of the chip are well defined.

For “Nature-Brain.png”, the 2-bit and 3-bit representations are completely erroneous in color scheme. This is likely due to the “nature-1.png” image being predominantly full of blue and gray tones. Also plotted in Figure 9 is the image remapped to $K = 128$ colors, corresponding to 7 bits, which I considered a good enough representation. With this bit-rate, enough of the greens can be extracted from “nature-1.png” such that they can better represent the original color scheme of “Nature-Brain.png”.

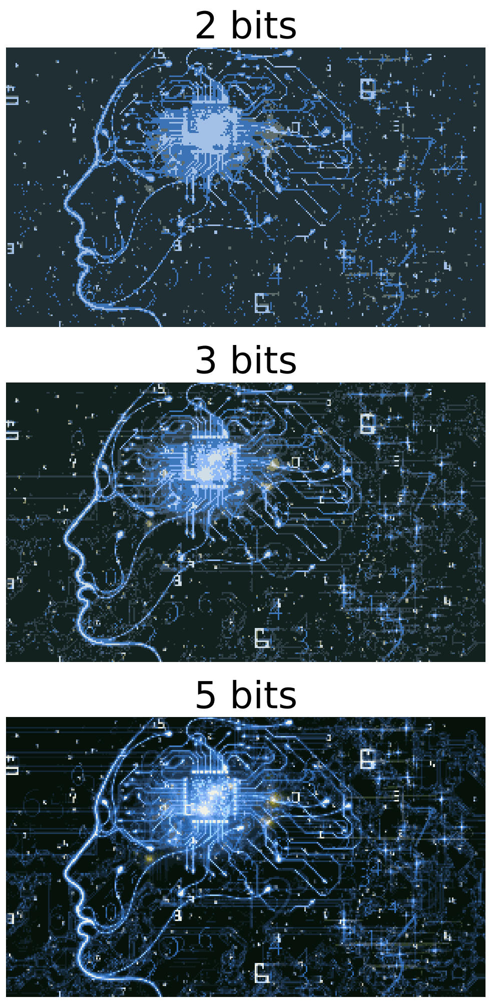


Fig. 8. The “machine-learning-1.png” image compressed to 2, 3 and 5 bits using the center colors of “nature-1.png”

E. Transferring compression centroids: From others to “nature-1.png”

The image compression is applied to the images “machine-learning-1.png” and “Nature-Brain.png”, and their center colors are used to remap the image “nature-1.png” to 6 bits, shown in Figure 10. With $K = 64$ colors, this was the bit-rate chosen that gives a good enough representation for using the centroids from both images.

Using the colors from “machine-learning-1.png”, the majority of the details and colors from the original “nature-1.png” image were recovered. This is due to both images consisting of large amounts of blues and grays. Where the remapped image

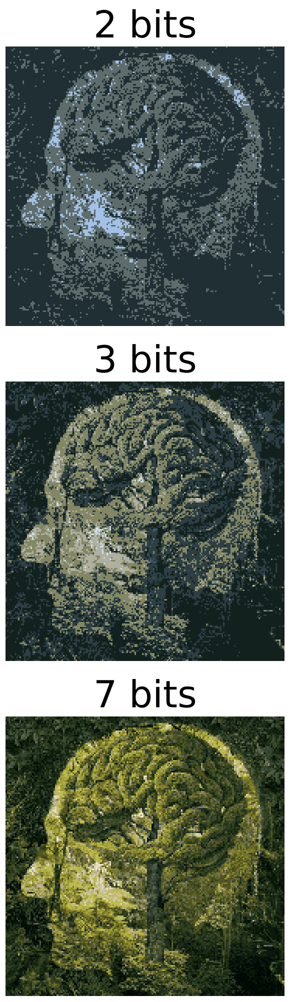


Fig. 9. The “Nature-Brain.png” image compressed to 2, 3 and 7 bits using the center colors of “nature-1.png”

falls short is in the trees, as the lack of greens prevents them from being accurately represented and are instead colored gray.

Using the colors from “Nature-Brain.png”, a good amount of detail can be obtained, but the color scheme is completely off. The lack of colors other than green in “Nature-Brain.png” make it impossible to represent the image in anything but a palette of greens. On the other hand, this allows the trees in the fore and background to be reconstructed in detail due to the variety of green tonalities available.

V. Problem 4: Polynomial Curve Fitting

A polynomial curve fitting routine is implemented in a class by the name “PolyFit” in Python and used in the following section.

A. Computing weights of “curvefitting.txt”

The data from the file “curvefitting.txt” is loaded and used to train for the optimal weights of polynomial fits of orders $M \in$

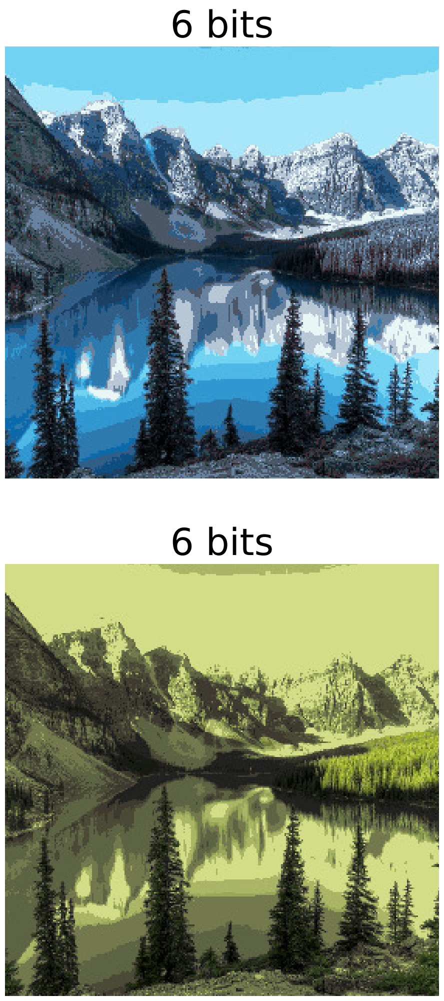


Fig. 10. The "nature-1.png" image compressed to 6 bits using the center colors of "machine-learning-1.png" (top) and "Nature-Brain.png" (bottom).

$\{0, 1, 3, 9\}$. The resulting weights $\boldsymbol{w}^*$ can be found in Table I, and the obtained models are plotted in Figure 11 alongside the data and the ideal sinusoidal function.

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
w_0^*	0.19	0.82	0.31	0.35
w_1^*		-1.27	7.99	232.36
w_2^*			-25.43	-5321.65
w_3^*			17.37	48566.75
w_4^*				-231632.39
w_5^*				640024.47
w_6^*				-1061772.84
w_7^*				1042374.51
w_8^*				-557669.94
w_9^*				125198.63

TABLE I
WEIGHTS OF POLYNOMIAL FITS OF VARIOUS ORDERS FOR THE "CURVEFITTING.TXT" DATA.

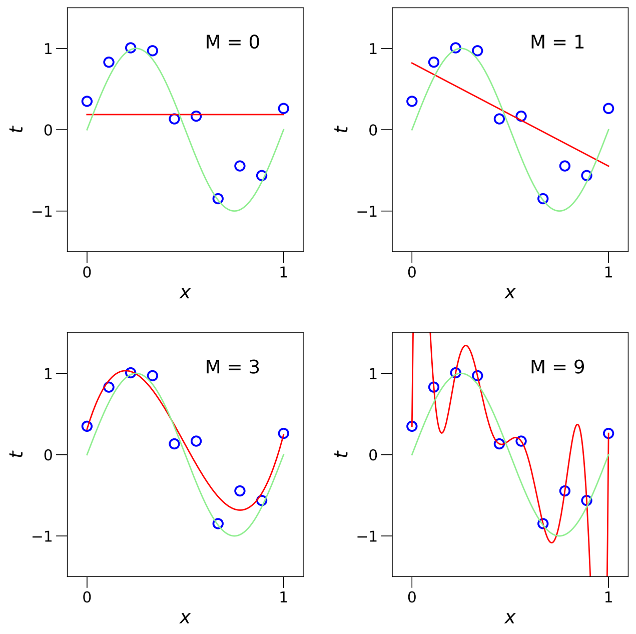


Fig. 11. Fitted polynomial models of orders 0, 1, 3 and 9 for the "curvefitting.txt" data.

B. Model error due to polynomial order

Using the "curvefitting.txt" as a training data set and a test data set of 50 points from a noisy sine function with a variance of $\sigma^2 = 0.09$, polynomial models of orders $M \in [0, 9]$ are fitted and their respective root-mean-square (rms) error E_{RMS} calculated, such that

$$E_{\text{RMS}} = \sqrt{\frac{1}{2N} \sum_{n=1}^{N} \left(t_n^{(\text{pred})} - t_n^{(\text{actual})} \right)^2} \tag{3}$$

with N being the number of data points and t being the predicted or actual outputs. The rms error for both training and testing data are plotted in Figure 12.

C. Model performance due to number of data points

A polynomial model of order $M = 9$ is fitted to two noisy sine data sets, one with number of data points $N = 15$ and the other with $N = 100$. Both with a variance of $\sigma^2 = 0.09$. The resulting fits are shown in Figure 13. There is less overfitting present as the number of data points increases.

VI. Problem 5: Polynomial Regression with Regularization

This section uses the previously mentioned PolyFit class as well, only with an added regularization term.

A. Computing weights of "curvefitting.txt"

Using the "curvefitting.txt" file as training data, two regularized polynomial models are trained on the data, with regularization terms of $\ln \lambda = -18$ and $\ln \lambda = 0$. The resulting weights $\boldsymbol{w}^*$ can be found in Table II and the models are plotted in Figure 14.

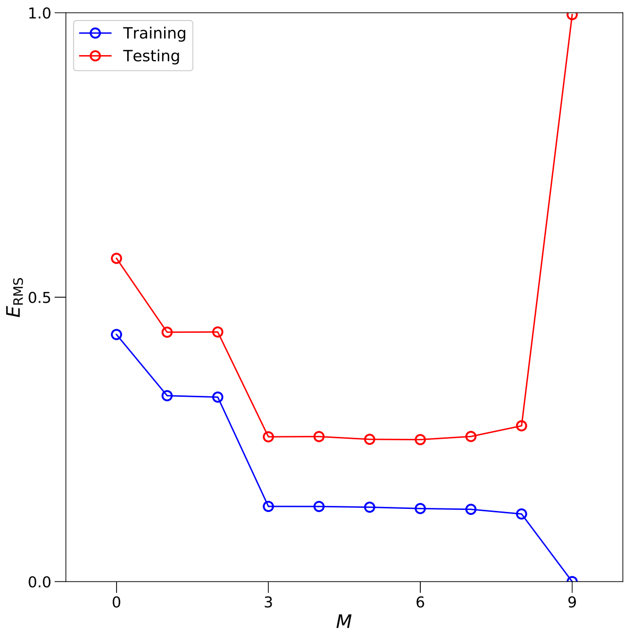
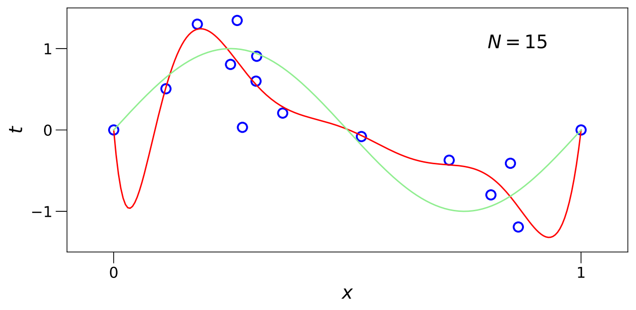


Fig. 12. Root-mean-square error of the training and testing data used for polynomial models of orders $M \in [0, 9]$.



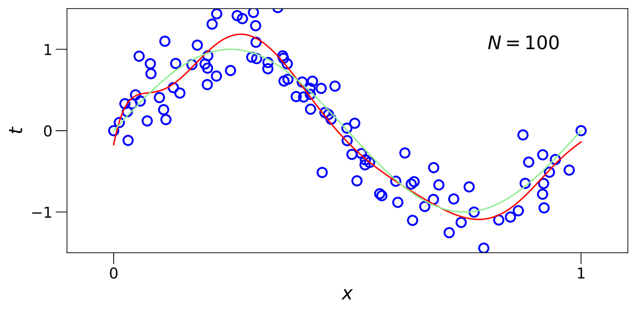


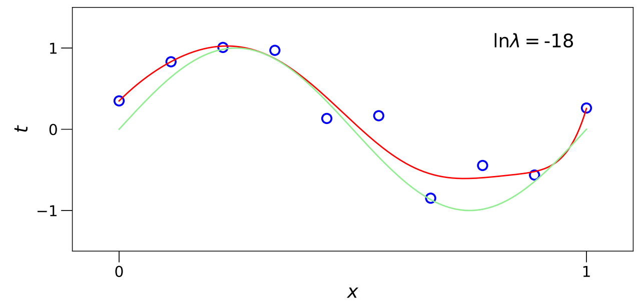
Fig. 13. Polynomial fits of order $M = 9$ for 15 and 100 data points of a noisy sine function with variance $\sigma^2 = 0.09$.

VII. Problem 6: K-Nearest Neighbors Classification

A K-Nearest Neighbors class by the name of KNN is implemented in Python and used in the following section to classify for various data sets. A meshgrid of varying size (depending on the data used) is created and the class for points within it is predicted considering $K \in \{1, 5, 10\}$ neighbors. A

	$\ln \lambda = -18$	$\ln \lambda = 0$
w_0^*	0.35	0.46
w_1^*	5.77	-0.34
w_2^*	-14.97	-0.37
w_3^*	29.86	-0.24
w_4^*	-95.62	-0.12
w_5^*	35.17	-0.02
w_6^*	168.71	0.05
w_7^*	-18.04	0.11
w_8^*	-273.18	0.16
w_9^*	162.19	0.20

TABLE II
WEIGHTS OF REGULARIZED POLYNOMIAL FITS OF VARIOUS ORDERS FOR THE "CURVEFITTING.TXT" DATA.



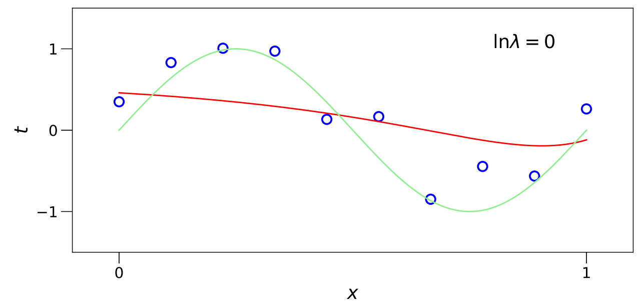


Fig. 14. Regularized polynomial fits of order $M = 9$ for the "curvefitting.txt" data with regularization terms $\ln \lambda = -18$ and $\ln \lambda = 0$.

contour plot is then generated to show the decision regions of the algorithm.

A. Concentric Gaussian Data Set Decision Regions

A concentric Gaussian data set with $N = 500$ data points, inner and outer variances $\sigma_{in}^2 = \sigma_{out}^2 = 1$, and outer annulus radius $r = 5$ is generated. The resulting decision regions for $K \in \{1, 5, 10\}$ neighbors is shown in Figure 15, alongside the original data set. As K increases, the shape of the inner circular Gaussian is better captured and approaches a circle.

B. Double Moon Data Set Decision Regions

A double moon data set with $N = 500$ data points, lower moon x-axis shift $d = 0$, radius $r = 1$ and width $w = 0.6$ is generated. The resulting decision regions for $K \in \{1, 5, 10\}$ neighbors is shown in Figure 16, alongside the original data set. As K increases, the overall shape stays consistent, with the

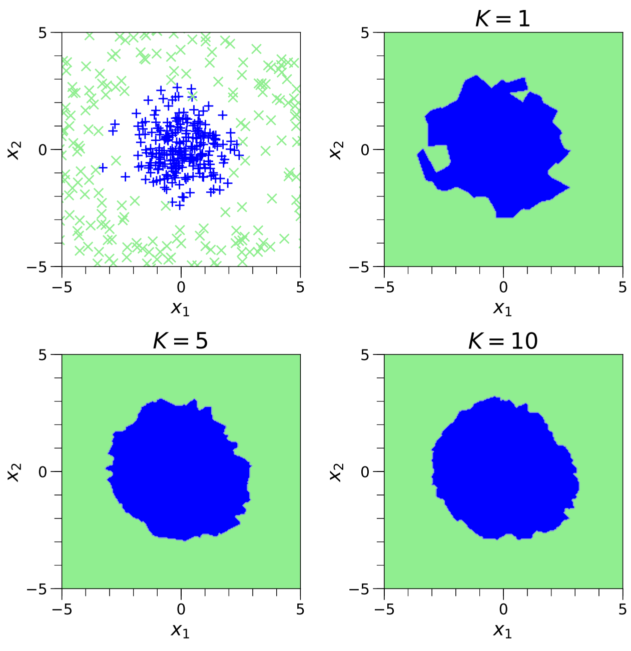


Fig. 15. Decision regions of K-Nearest Neighbors for a concentric Gaussian data set with $K = 1$, $K = 5$ and $K = 10$.

sides after the curves (around the $(-0.5, -0.5)$ and $(1.75, 0.5)$ points) getting smoother and straighter with higher values of K.

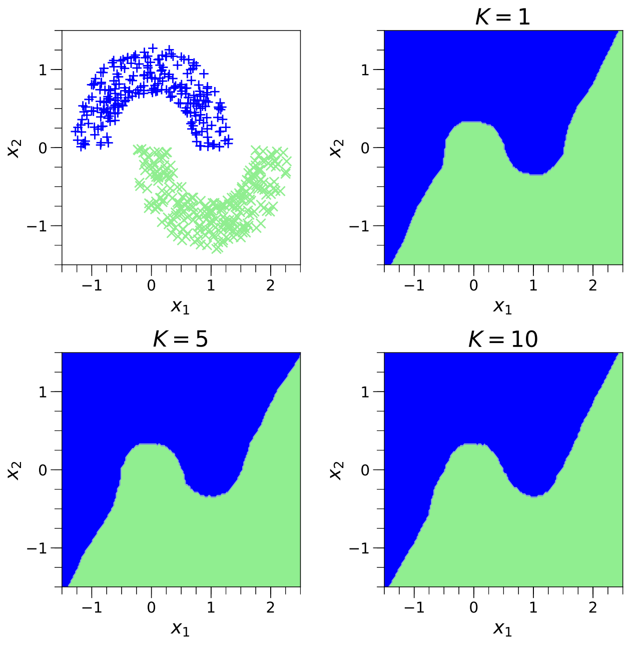


Fig. 16. Decision regions of K-Nearest Neighbors for a double moon data set with $K = 1$, $K = 5$ and $K = 10$.

C. Gaussian XOR Data Set Decision Regions

A Gaussian XOR data set with $N = 500$ and variance $\sigma^2 = 1$ is generated. The resulting decision regions for $K \in \{1, 5, 10\}$ neighbors is shown in Figure 17, alongside the original data set. As K increases, the decision region stays fairly consistent. Deviations from the ideal straight boundary are dependant on the density of outlier data points.

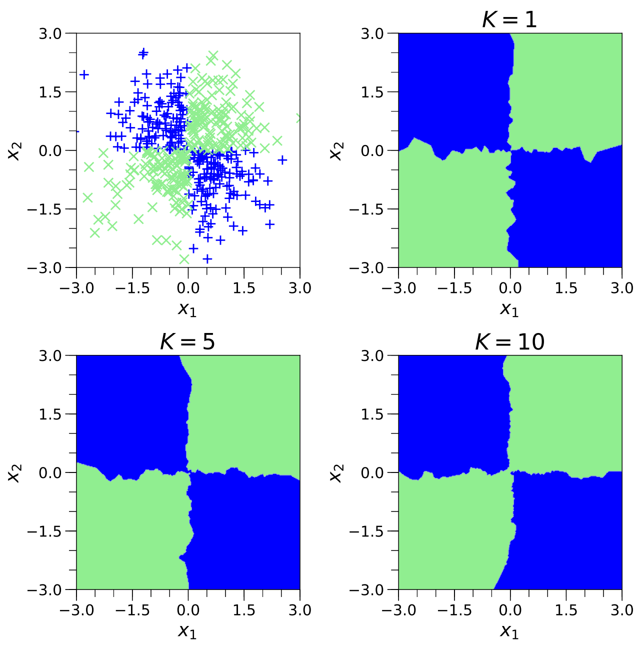


Fig. 17. Decision regions of K-Nearest Neighbors for a Gaussian XOR data set with $K = 1$, $K = 5$ and $K = 10$.

VIII. Conclusion

In this project, we performed clustering, regression and classification tasks using K-means, polynomial curve fitting and K-Nearest Neighbors algorithms accordingly. We also applied K-means to a real-world and applicable problem to demonstrate potential uses of unsupervised learning algorithms.