IAML – INFR10069 (LEVEL 10): Assignment #1

s1824086

Question 1 : (22 total points) Linear Regression

In this question we will fit linear regression models to data.

(a) (3 points) Describe the main properties of the data, focusing on the size, data ranges, and data types.

The data set contains 50 observations, i.e. rows, each row corresponding to one student. There are 2 attributes in the data, both numerical (real-valued), one of them corresponding to the amount of time a student spent studying for an exam, the other one corresponding to the score a student reached in the exam. Revision time values range from 2.7 to 48.0 (hours), whereas the observed exam results range from 14.7 to 94.9 (marks). The according mean values lie at 22.2(h) and 49.9(marks), respectively. There appear to be no missing values in the data.

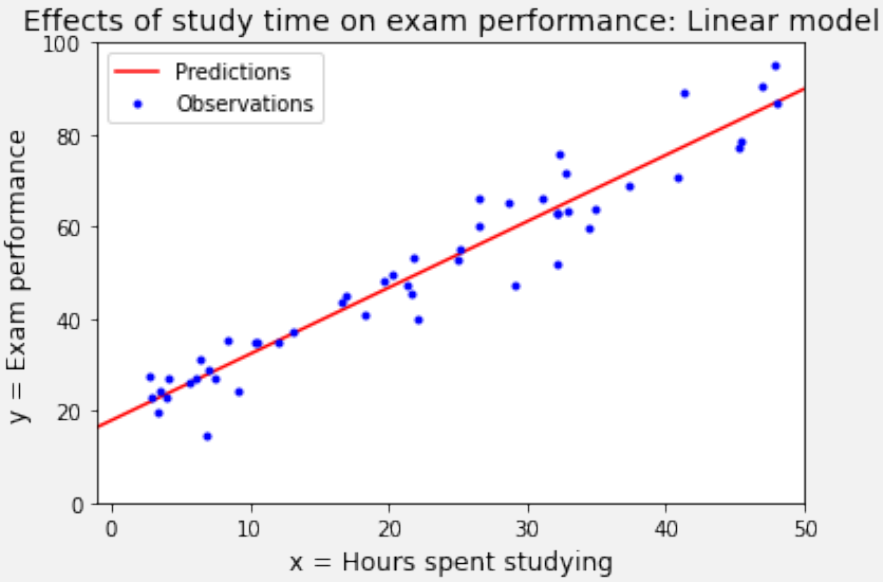
Question 1 continued on next page...

(b) (3 points) Fit a linear model to the data so that we can predict `exam_score` from `revision_time`. Report the estimated model parameters $\mathbf{w}$. Describe what the parameters represent for this 1D data. For this part, you should use the sklearn implementation of Linear Regression.
Hint: By default in sklearn `fit_intercept = True`*. Instead, set* `fit_intercept = False` *and pre-pend* 1 *to each value of* x_i *yourself to create* $\boldsymbol{\phi}(x_i) = [1, x_i]$.

Our estimator spits out weights $\mathbf{w} = [17.8977, 1.4411]$. This means that our model will estimate an exam score of 17.8977 + 1.4411*x for a student, given the piece of information that the student studied x hours for the exam.
I.e. a student that did not study at all would be estimated to reach 17.8977 marks, and any student would be estimated to reach roughly 1.4411 marks more than that, per hour that they did spend studying. Geometrically speaking, 17.8977 represents the y-intercept, 1.4411 the slope of a linear univariate function.

Question 1 continued on next page...

(c) (3 points) Display the fitted linear model and the input data on the same plot.



Question 1 continued on next page...

(d) (3 points) Instead of using sklearn, implement the closed-form solution for fitting a linear regression model yourself using numpy array operations. Report your code in the answer box. It should only take a few lines (i.e. <5).
Hint: Only report the relevant lines for estimating $\mathbf{w}$ *e.g. we do not need to see the data loading code. You can write the code in the answer box directly or paste in an image of it.*

The closed-form solution for fitting a linear regression model is called "the Normal Equation". It is of the form $\hat{\mathbf{w}} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{y}$.

In Python:

```
In [20]:
# Phi is our nx2 design matrix, y our nx1 target vector
gramian_matrix_of_phi = np.dot(Phi.T, Phi)
inverse_of_gramian    = np.linalg.inv(gramian_matrix_of_phi)
pseudo_inverse_of_phi = np.dot(inverse_of_gramian, Phi.T)
estimated_weights     = np.dot(pseudo_inverse_of_phi, y)
estimated_weights

Out[20]: array([17.89768026,  1.44114091])
```

Question 1 continued on next page...

(e) (3 points) Mean Squared Error (MSE) is a common metric used for evaluating the performance of regression models. Write out the expression for MSE and list one of its limitations.
Hint: For notation, you can use y for the ground truth quantity and $\hat{y}$ (`$\hat{y}$` in latex) in place of the model prediction.

Given our model's prediction function h, as well as m feature vectors $\mathbf{x}^i$, composing feature matrix $\mathbf{X}$, we shall write $\hat{y}_i := h(\mathbf{x}^i)$ for our model's prediction of the target value y_i. Then, $MSE(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y^{(i)})^2$.
MSE has one big drawback: It is very sensitive to outliers, which can lead to unintuitive evaluations. E.g., suppose m=100. If 99 of our model's predictions are on point, but one of them is off by 10, $MSE(\mathbf{X}, h) = 1$. MSE would take the same value if all 100 predictions were off by 1, which may not be what we want.

Question 1 continued on next page...

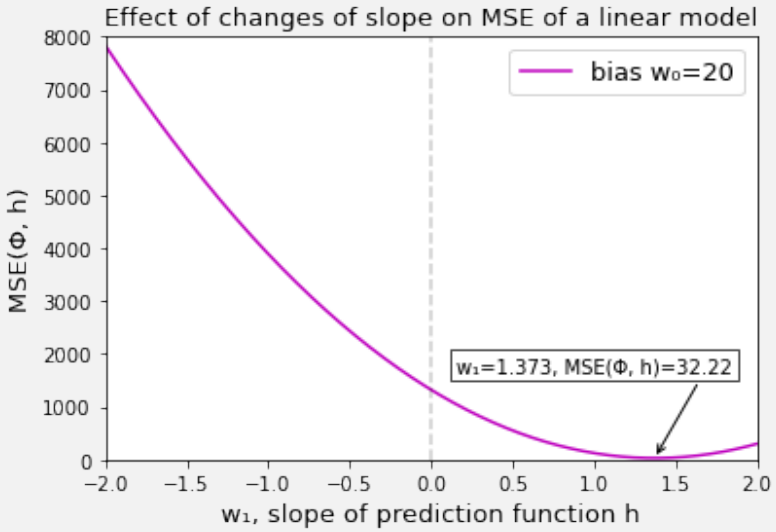
(f) (3 points) Our next step will be to evaluate the performance of the fitted models using Mean Squared Error (MSE). Report the MSE of the data in `regression_part1.csv` for your prediction of `exam_score`. You should report the MSE for the linear model fitted using sklearn and the model resulting from your closed-form solution. Comment on any differences in their performance.

The MSE for the sklearn LinearRegression() model is roughly 30.9855, as computed using sklearn's mean_squared_error() function.
The manually computed MSE for my closed-form solution yields the exact same result, rounded to 13 significant digits.
This shouldn't startle us, since it simply confirms that we have done a good job at applying the closed-form solution.

Question 1 continued on next page...

(g) (4 points) Assume that the optimal value of w_0 is 20, it is not but let's assume so for now. Create a plot where you vary w_1 from -2 to $+2$ on the horizontal axis, and report the Mean Squared Error on the vertical axis for each setting of $\mathbf{w} = [w_0, w_1]$ across the dataset. Describe the resulting plot. Where is its minimum? Is this value to be expected? *Hint: You can try 100 values of* w_1 *i.e.* `w1 = np.linspace(-2,2, 100)`.

We are fixing $w_0 = 20$, as opposed to the previously computed value of roughly 17.9. Since the y-intercept of our prediction function is therefore slightly higher than before, we would then expect a slope that's slightly lower than the previously computed 1.44 to lead to a minimal MSE. Afterall, if it had the same slope as before, its estimations would tend to be slightly too large. Also, we would expect the minimal MSE we can find this way to be slightly greater than 30.9855, since we are not using the ideal y-intercept we computed earlier.

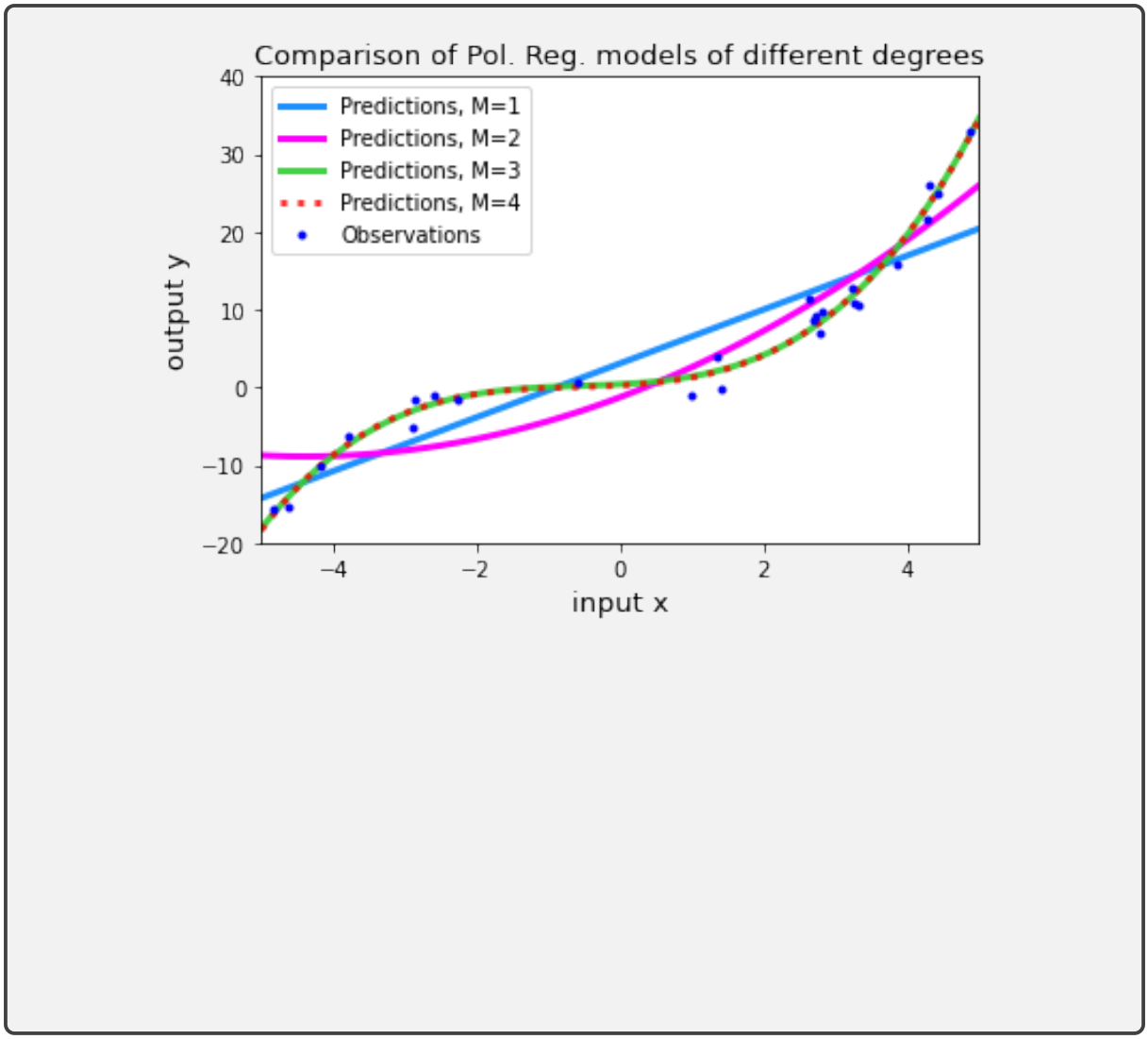


Indeed, we observe the lowest MSE for a slope of 1.373, with a mean squared error of 32.22. We do not observe any other local minima, i.e. the MSE gets worse the further we stray from $w_1 = 1.373$. Furthermore, the rate at which the MSE gets worse rises the further we stray from this minimum. The plot resembles a cutout of a parabola with a vertical axis of symmetry.

Question 2 : (18 total points) Nonlinear Regression

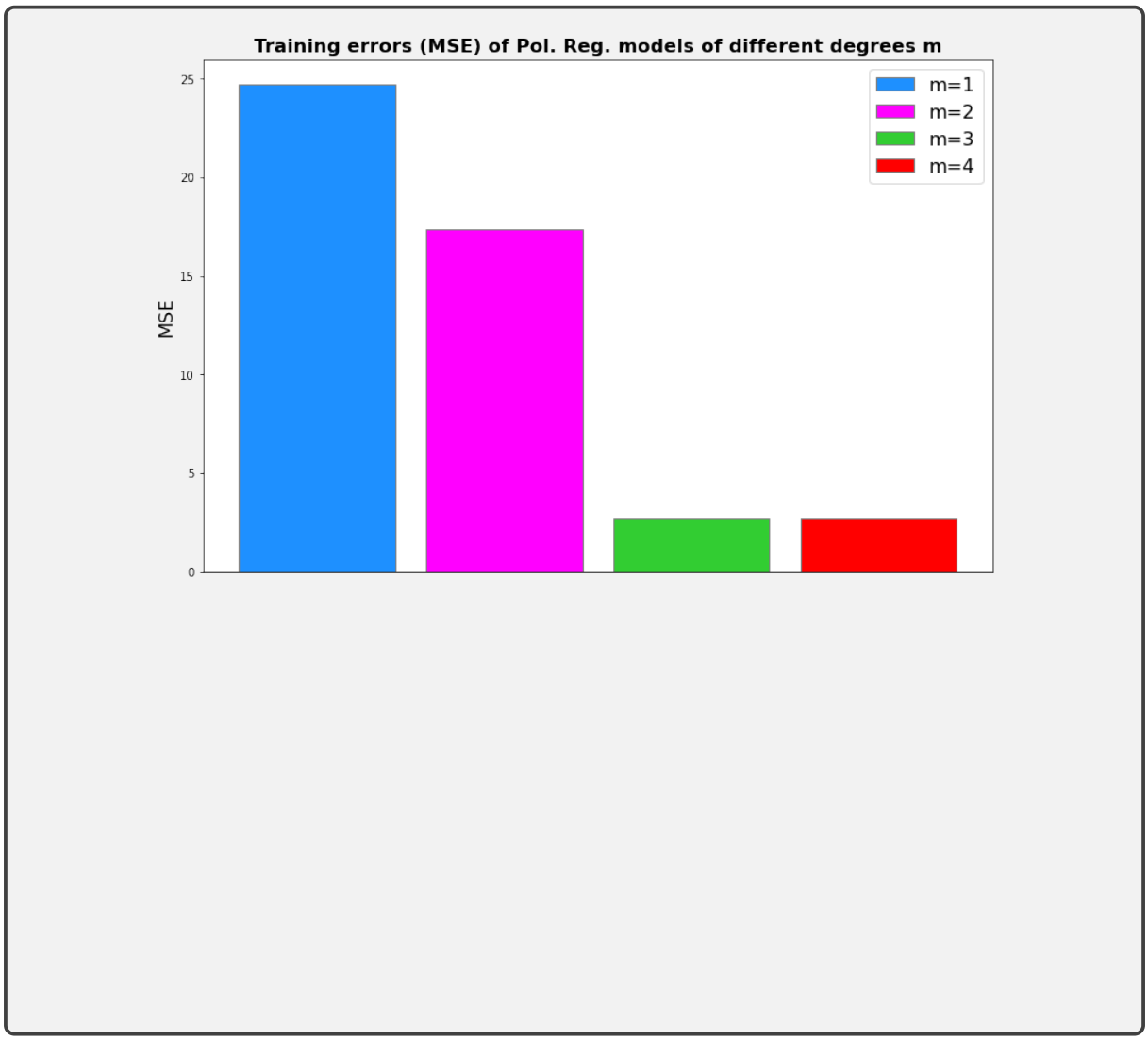
In this question we will tackle regression using basis functions.

(a) (5 points) Fit four different polynomial regression models to the data by varying the degree of polynomial features used i.e. $M = 1$ to 4. For example, $M = 3$ means that $\boldsymbol{\phi}(x_i) = [1, x_i, x_i^2, x_i^3]$. Plot the resulting models on the same plot and also include the input data.
Hint: You can again use the sklearn implementation of Linear Regression and you can also use PolynomialFeatures to generate the polynomial features. Again, set `fit_intercept = False`.



Question 2 continued on next page...

(b) (3 points) Create a bar plot where you display the Mean Squared Error of each of the four different polynomial regression models from the previous question.



Question 2 continued on next page...

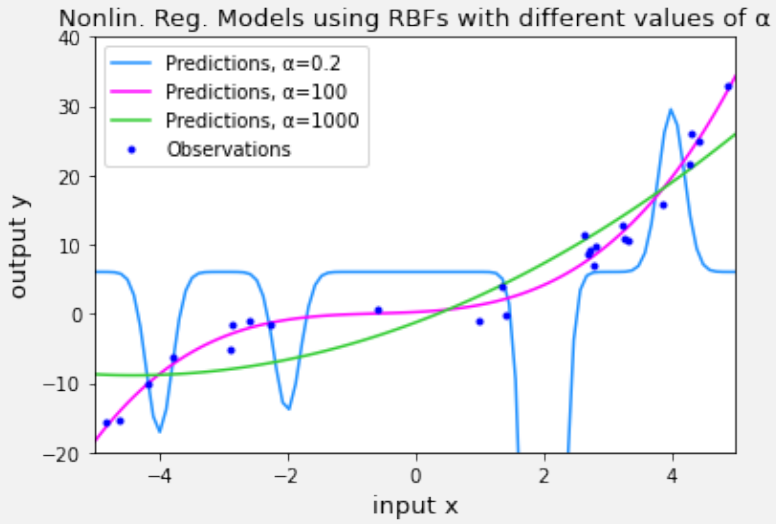
(c) (4 points) Comment on the fit and Mean Squared Error values of the $M = 3$ and $M = 4$ polynomial regression models. Do they result in the same or different performance? Based on these results, which model would you choose?

We notice that there is a strong overlap between the graphs generated by the pol. reg. models of degrees 3 and 4. Consequently, the training error of the quartic model is only marginally smaller than the one of the cubic model.
When two models perform equally well on the training data, it is generally wise to choose the less complex model. Therefore, I would choose the cubic model in this example, since a degree of 3 seems to be sufficient to achieve a good fit to the data, whereas further increasing the degree to 4 achieves very little.

(Note: Because the graphs are so similar, we actually wouldn't expect model 4 to do (much) worse on unseen data than model 3, which often happens to models that are too complex. Choosing the simpler model is good practice nevertheless.)

Question 2 continued on next page...

(d) (6 points) Instead of using polynomial basis functions, in this final part we will use another type of basis function - radial basis functions (RBF). Specifically, we will define $\boldsymbol{\phi}(x_i) = [1, rbf(x_i; c_1, \alpha), rbf(x_i; c_2, \alpha), rbf(x_i; c_3, \alpha), rbf(x_i; c_4, \alpha)]$, where $rbf(x; c, \alpha) = \exp(-0.5(x - c)^2/\alpha^2)$ is an RBF kernel with center c and width α. Note that in this example, we are using the same width α for each RBF, but different centers for each.
Let $c_1 = -4.0$, $c_2 = -2.0$, $c_3 = 2.0$, and $c_4 = 4.0$ and plot the resulting nonlinear predictions using the `regression_part2.csv` dataset for $\alpha \in \{0.2, 100, 1000\}$. You can plot all three results on the same figure. Comment on the impact of larger or smaller values of α.



Looking at the definition of an individual RBF, we see that choosing a small α will make the peak of its graph thin, whereas a large α will make it wide. When we use multiple RBFs with a fixed α to transform our data, choosing a greater α thus leads to each RBF having an effect on predictions based on x values from a wider range.
In our case, choosing $\alpha = 0.2$ leads to our 4 RBFs having separate areas of effect, with spaces between them. Predictions for x-values lying very close to one of the 4 centers are heavily pulled in one direction, whereas all other predictions lie at the mean. For α=100 we can imagine the areas of effect of the RBFs overlapping, however we still notice the slope of the line changing more rapidly around the centers of our RBFs, achieving a tight fit. For α=1000 this is less the case: The graph looks like it has been smoothed out, resulting in a less tight fit to the training data.

Question 3 : (26 total points) Decision Trees

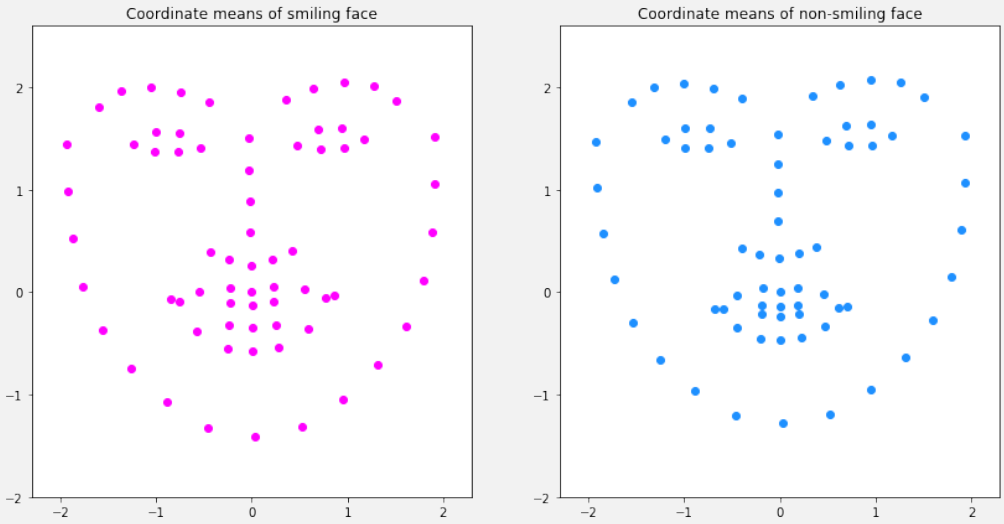
In this question we will train a classifier to predict if a person is smiling or not.

(a) (4 points) Load the data, taking care to separate the target binary class label we want to predict, `smiling`, from the input attributes. Summarise the main properties of both the training and test splits.

Both datasets feature 136 real-valued input attributes, describing 68 2D coordinates, and a binary target attribute capturing whether the according person was smiling at the time their face was analyzed. We have 4800 instances in the training, 1200 in the testing set, meaning that an 80-20-split was chosen. All attributes lie primarily in the range (-3, 3) (different attributes being centered around / confined to different parts of this interval) and there are no outliers outside of the range (-4, 4). 48.65% of people are smiling in the training, 49.33% in the testing set, which means that our classes are relatively balanced.

Question 3 continued on next page...

(b) (4 points) Even though the input attributes are high dimensional, they actually consist of a set of 2D coordinates representing points on the faces of each person in the dataset. Create a scatter plot of the average location for each 2D coordinate. One for (i) smiling and (ii) one not smiling faces. For instance, in the case of smiling faces, you would average each of the rows where `smiling = 1`. You can plot both on the same figure, but use different colors for each of the two cases. Comment on any difference you notice between the two sets of points.
Hint: Your plot should contain two faces.



The main difference between the two scatter plots is that, as opposed to the "average non-smiling person", the corners of mouth of the "average smiling person" point upwards, and that there is a small gap between the upper and lower lip. Furthermore, the mouth of a smiling person tends to be wider. The nose of the average smiling person looks a tiny bit different as well, presumably caused by muscles that pull the nasal tip down and the nostrils laterally, but there are no other obvious differences between the two plots. Squinting, which may perhaps help indicate a smile in some cases, could possibly be captured by our 2D coordinate representation, but if it is, this is not apparent from looking at the 2 plots. Some other possibly useful features, like smile lines, are not captured by the representation.

Question 3 continued on next page...

(c) (2 points) There are different measures that can be used in decision trees when evaluating the quality of a split. What measure of purity at a node does the DecisionTreeClassifier in sklearn use for classification by default? What is the advantage, if any, of using this measure compared to entropy?

By default, Gini impurity is used to measure the impurity at a node of our tree. The i^{th} node has Gini score $G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$, a Gini score of 0 thus corresponding to a pure node. We sum over all classes k, where $p_{i,k}$ is the ratio of instances of class k in node i. The difference to the formula for entropy ($H_i = -\sum_{k=1, p_{i,k} \neq 0}^{n} p_{i,k} log_2(p_{i,k})$) is quite small, but in it lies the advantage of using Gini impurity over entropy: It is slightly faster to compute because no log is involved. However, they usually lead to very similar trees.

Question 3 continued on next page...

(d) (3 points) One of the hyper-parameters of a decision tree classifier is the maximum depth of the tree. What impact does smaller or larger values of this parameter have? Give one potential problem for small values and two for large values.

Choosing a max. depth of n means that each subset of our data at a leaf of our tree resulted from at most n splits. A larger max. depth may lead to "overfitting" the training data, i.e. capturing patterns that are merely caused by random noise, and a model that does not generalize well to unseen data. It also makes it harder for humans to understand the model. On the other hand, small values may not allow our algorithm to make enough splits. Our model will not capture patterns in the data well enough to do well on either training or unseen data.

Question 3 continued on next page...

(e) (6 points) Train three different decision tree classifiers with a maximum depth of 2, 8, and 20 respectively. Report the maximum depth, the training accuracy (in %), and the test accuracy (in %) for each of the three trees. Comment on which model is best and why it is best.
Hint: Set `random_state = 2001` *and use the* `predict()` *method of the DecisionTreeClassifier so that you do not need to set a threshold on the output predictions. You can set the maximum depth of the decision tree using the* `max_depth` *hyper-parameter.*

The tree with a max. depth of 2 has a training accuracy of 79.5% and a testing accuracy of 78.2%. The tree with a max.depth of 8 has a training accuracy of 93.4% and a testing accuracy of 84.1%, whereas the one with a max. depth of 20 has perfect training accuracy and a testing accuracy of 81.5%.
Choosing the tree with a depth of 2 would be bad, because it neither performs well on the training, nor on the testing data. Thus, it is underfitting, showing us that a depth of 2 is not enough to capture what divides the data. The tree with a depth of 8 has a slightly better performance on the testing data than the one with a depth of 20. It would be unwise to choose the tree with depth 20, given that a less complex model achieves a better testing accuracy (which seems like a valid metric in this context, because our 2 classes are balanced). It is therefore overfitting. Its perfect accuracy on the testing data is nothing special- if we have no duplicates in our data, a decision tree with sufficient depth will always fit the training data perfectly.
Thus, out of the 3 options, the tree with a depth of 8 is clearly the best one. We have found the golden middle between underfitting and overfitting.

Question 3 continued on next page...

(f) (5 points) Report the names of the top three most important attributes, in order of importance, according to the Gini importance from DecisionTreeClassifier. Does the one with the highest importance make sense in the context of this classification task?
Hint: Use the trained model with `max_depth = 8` *and again set* `random_state = 2001`.

The most important attribute is x50, followed by y48 and y29. Attribute x50 corresponds to the x-position of the leftmost of the 3 "central points" on top of the upper lip. This is perhaps a bit surprising. I would have expected the y-values of the corners of the mouth (e.g. y48) to be more indicative of whether a person is smiling. But since the mouth does get wider when we smile, it does make sense that this attribute is deemed important.

Question 3 continued on next page...

(g) (2 points) Are there any limitations of the current choice of input attributes used i.e. 2D point locations? If so, name one.

There are certain downsides to this choice. For example, our model would not be able to make sense of any depiction of a face at an angle. Also, any appropriate input data would need to be centered, and rotated images prohibited, for our model to make sense of the individual coordinates. Therefore, our choice of attributes puts some rather heavy restrictions on the data our model is capable of dealing with.

Question 4 : (14 total points) Evaluating Binary Classifiers

In this question we will perform performance evaluation of binary classifiers.

(a) (4 points) Report the classification accuracy (in %) for each of the four different models using the `gt` attribute as the ground truth class labels. Use a threshold of $>= 0.5$ to convert the continuous classifier outputs into binary predictions. Which model is the best according to this metric? What, if any, are the limitations of the above method for computing accuracy and how would you improve it without changing the metric used?

For models 1 to 4 we observe accuracy scores of 61.6, 55, 32.1 and 32.9 %, respectively, i.e. model 1 is the best according to this metric.
Even if we decide that accuracy is the metric we want to use (it depends on the context whether this is appropriate), there is a problem with the above approach: We simply assumed that 0.5 is the threshold we should use, which may not be true. Each model may perform best with a different threshold and therefore, choosing one threshold for all of them does not lead to a fair comparison and does not allow us to pick which model is best. An easy way of addressing this would be to find out which threshold yields the best accuracy for each model, before comparing the models by only looking at the best accuracy score each model achieved.

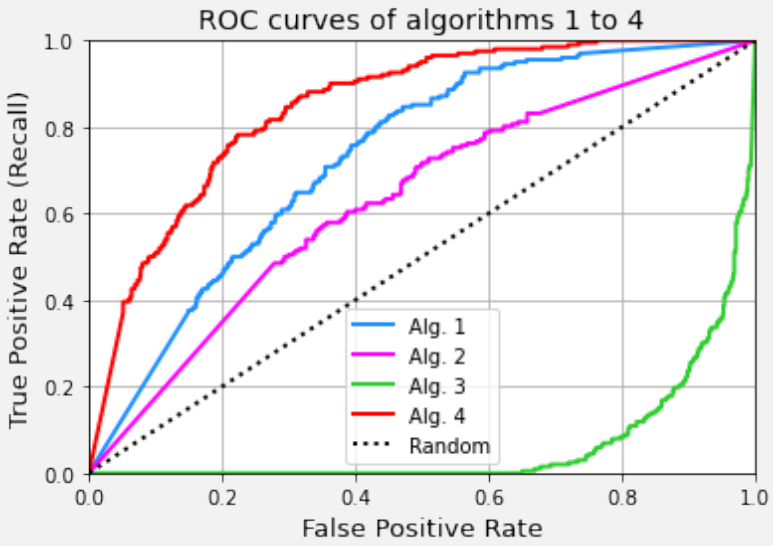
Question 4 continued on next page...

(b) (4 points) Instead of using classification accuracy, report the Area Under the ROC Curve (AUC) for each model. Does the model with the best AUC also have the best accuracy? If not, why not?
Hint: You can use the roc_auc_score function from sklearn.

For models 1 to 4 we observe AUCs of 0.7321, 0.6316, 0.0640 and 0.8474. Thus, the fourth algorithm has the best AUC. I.e. algorithm 1 suddenly gets outperformed by algorithm 4, despite having been almost twice as accurate for a threshold of 0.5. For this threshold, algorithm 4 had a perfect recall of 1, but only a precision of 0.23. Its mean output on the testing data lies at 0.74, even though only 20.2% of data points in the set belong to class 1. This caused algorithm 4 to falsely assign many data points to class 1 - but the ones it assigned to class 0 were actually in that class. Algorithm 4 can achieve a good AUC, because raising the threshold, thereby rapidly decreasing the ratio of "0" instances that are incorrectly classified as "1"s, lowers its recall relatively slowly. This shows that the accuracy of a model (especially for one specific threshold) teaches us only little about its AUC.

Question 4 continued on next page...

(c) (6 points) Plot ROC curves for each of the four models on the same plot. Comment on the ROC curve for `alg_3`? Is there anything that can be done to improve the performance of `alg_3` without having to retrain the model?
Hint: You can use the roc_curve function from sklearn.



The ROC curve of algorithm 3 lies below even our "random classifier" baseline for any threshold we might choose. The algorithm's recall drops to 0 if we choose a threshold that leads to a False Positive Rate of 60% or less. This sounds ridiculously bad, but we can make the model useful by simply "inverting" it. We define algorithm 5 s.t. it assigns output $p_5 = (1 - p_3)$ to input x, where p_3 is the output algorithm 3 assigns to x. We will have $AUC_5 \approx 1 - AUC_3$. And why is that?
$p_3 > t \Leftrightarrow p_5 < 1 - t$. Excluding the special case where $p_3 = t$, this means that FPR_3 and TPR_3 for threshold t equal $TNR_5 = 1 - FPR_5$ and $FNR_5 = 1 - TPR_5$, respectively, for threshold $1 - t$. Thus, since thresholds range from 0 to 1, for each point (a,b) on the ROC curve of alg. 3 we have point (1-a,1-b) on the ROC curve of alg. 5. Our claim follows. In our example, $AUC_5 = 0.9360$.