# IAML – INFR10069 (LEVEL 10): Assignment #1

s1703728

# 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 comprises of 50 entries, a 1 dimensional input $'revision\_time'$ and a 1 dimensional output, $'exam\_score'$. The data type for both is $float64$.
For the $'revision\_time'$, the minimum value is 2.723, the maximum value is 48.011, and the mean is 22.220 with standard deviation 13.896.
For the $'exam\_score'$, the minimum value is 14.731, the maximum value is 94.945, and the mean is 49.920 with standard deviation 20.026.

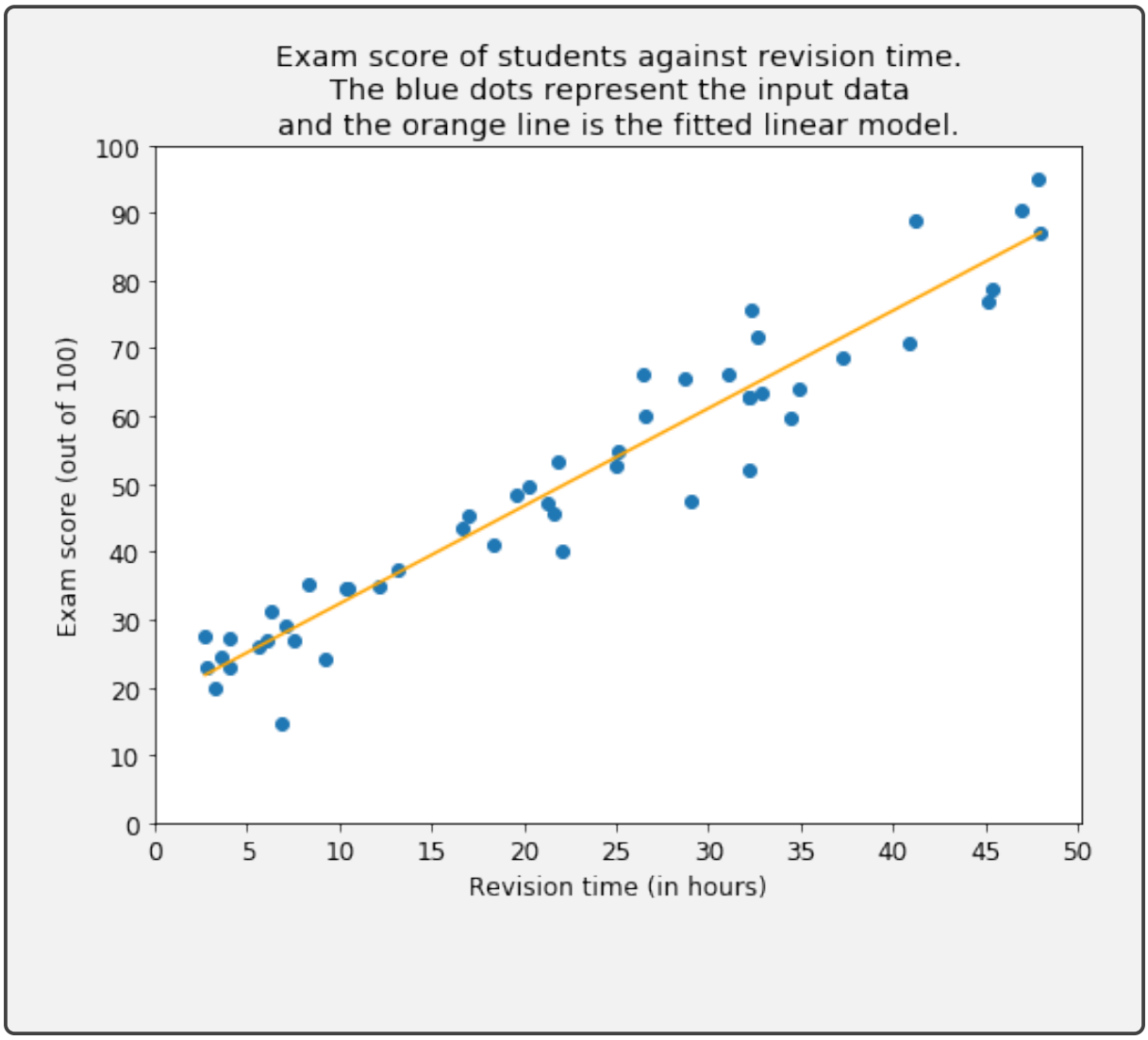
*Question 1 continued on next page...*

(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* $\boldsymbol{\phi}(x_i) = [1, x_i]$.

In this case, our $\mathbf{w}$ contains 2 components, $w_0$ and $w_1$. The estimated model parameters to 3 decimal places are: $\mathbf{w} = \begin{bmatrix} 17.898 \\ 1.441 \end{bmatrix}$
In the case of this linear model for this 1D data, the first corresponds to the y-axis intercept, and the second corresponds to the gradient of the straight line.

*Question 1 continued on next page...*

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



(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* **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 code for estimating **w**:

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(1)
phi = poly.fit_transform(X)
inverse = np.linalg.inv( (phi.T).dot(phi) )
w = ( inverse.dot(phi.T) ).dot(y)
```

The code for calculating the output predictions:

```
y_pred = phi.dot(w)
```

*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.*

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

where N is the total number of data entries (rows).

One of its limitations is that it is sensitive to outliers. One wrong entry (maybe typo or any mistake) would largely affect the value of the MSE.

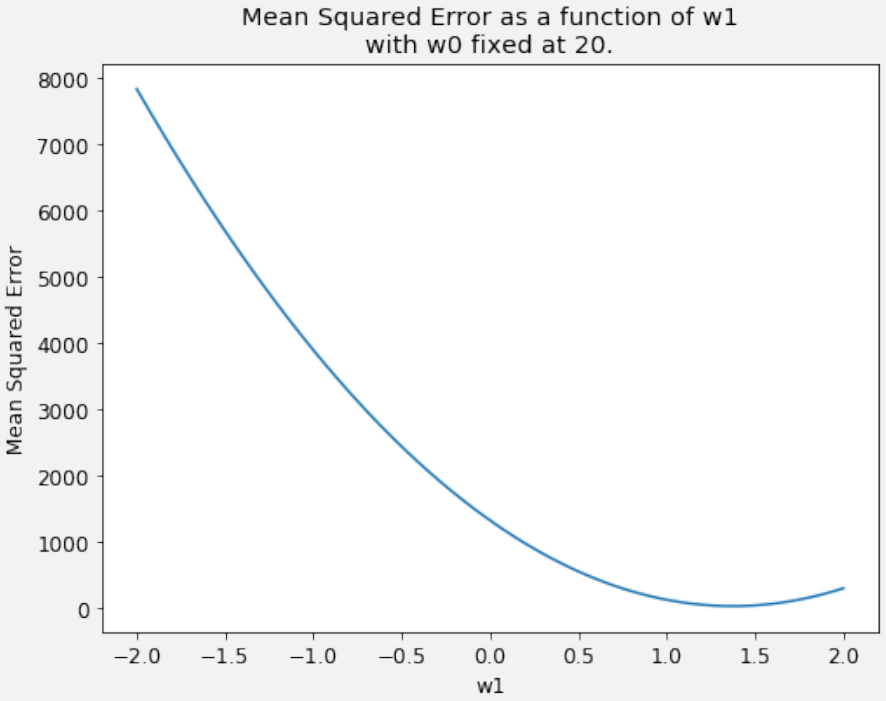
*Question 1 continued on next page...*

(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 Mean Squared Error for the linear model fitted using *sklearn* was 30.985. The Mean Squared Error for the model resulting from the closed-form solution was 30.985. Both were given to 3 decimal places.
These two values are identical, which means that the two models perform equally well.

*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)`.



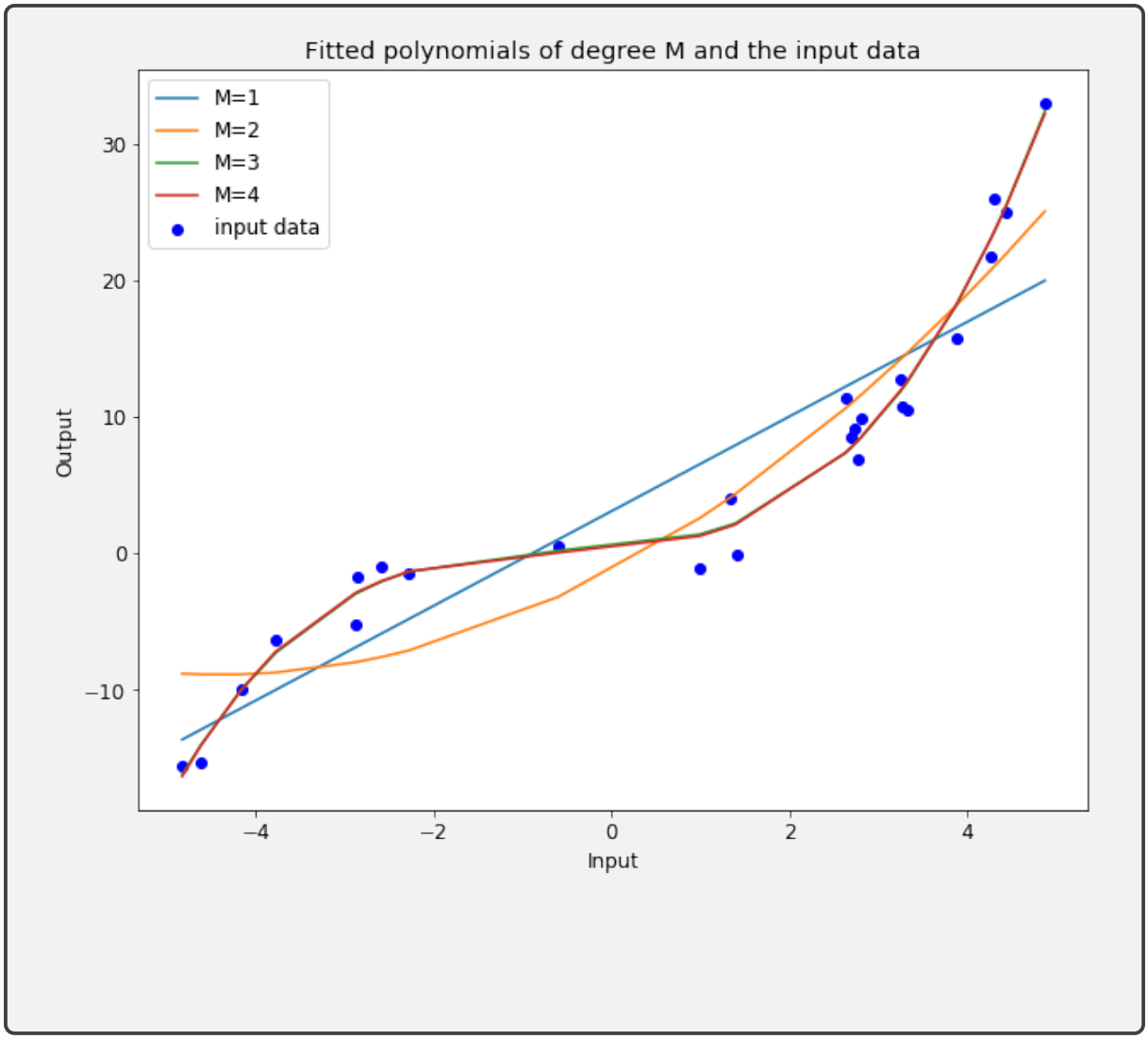

The resulting plot above shows the MSE as $w_1$ is varied from -2 to $+2$. The relationship observed is that initially the error decreases as $w_1$ increases, but only until it reaches a minimum at around 1.4, where it starts increasing again. Given that our $w_0$ was fixed at 20, that value was expected, since the $\mathbf{w}$ I got from implementing the closed form solution in part $(d)$ was $\mathbf{w} = \begin{bmatrix} 17.898 \\ 1.441 \end{bmatrix}$, so I expected something around 1.4 in this case. The reason is that in Linear Regression the model is produced by minimising the MSE.

# Question 2 : (18 total points) Nonlinear Regression

**In this question we will tackle regression using basis functions.**

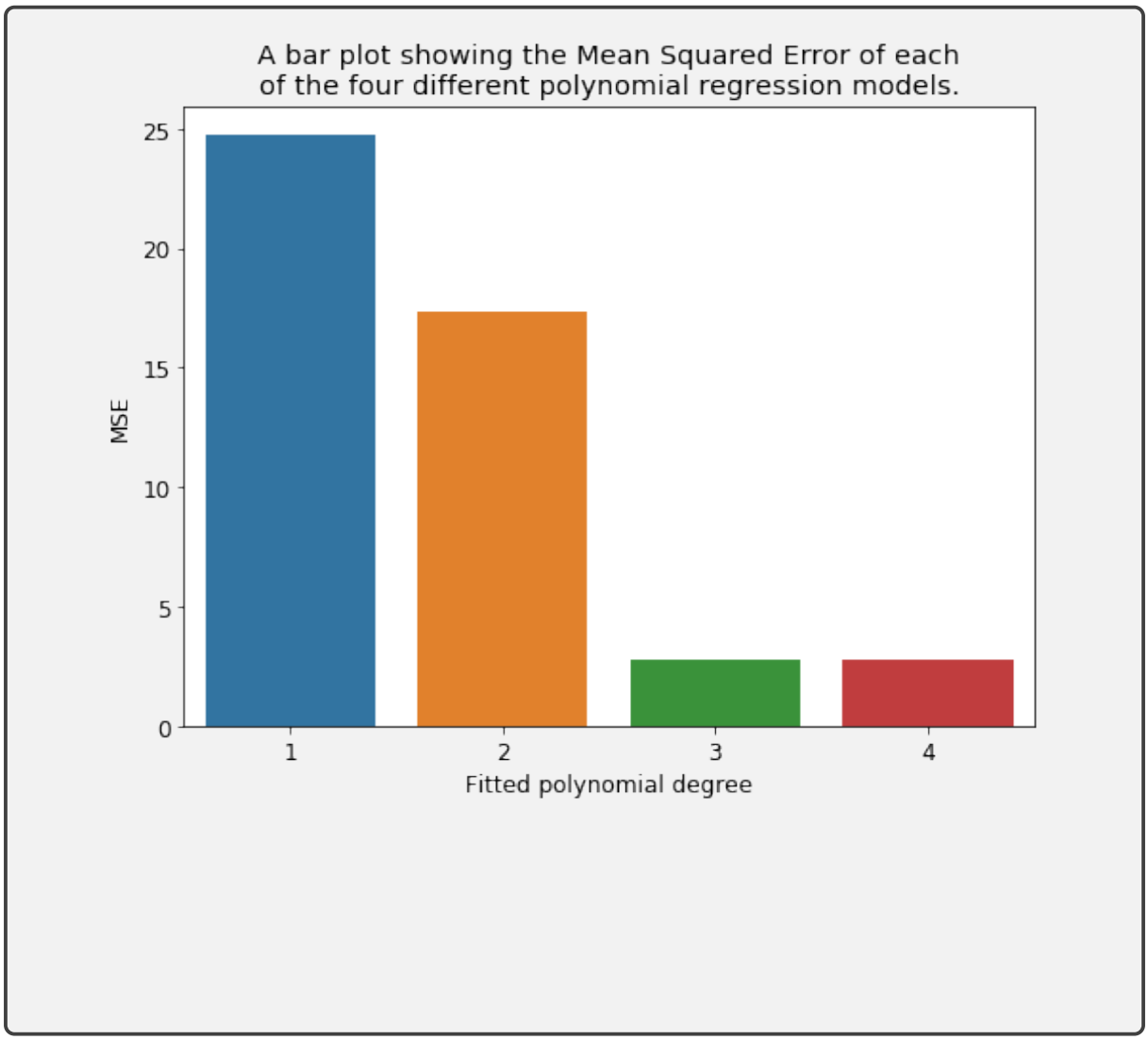
(a) (5 points) Fit four different polynomial regression models to the data by varying the degree of polynomial features used i.e. $M = 1$ to 4. For example, $M = 3$ means that $\boldsymbol{\phi}(x_i) = [1, x_i, x_i^2, x_i^3]$. Plot the resulting models on the same plot and also include the input data.

*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.



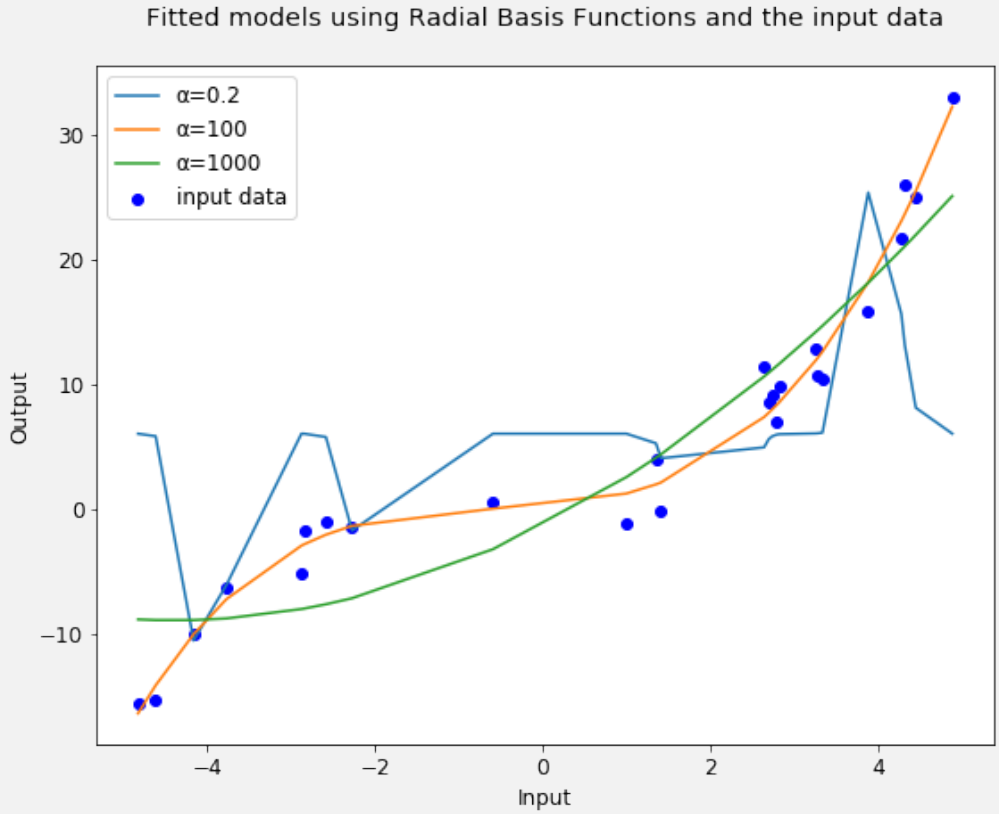

*Question 2 continued on next page...*

(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?

The $M = 3$ and $M = 4$ polynomial regression models appear to have the same fit in the plot in part $(a)$ and their MSE values are the same within 3 significant figures (both are equal to 2.74). Therefore, I would say that they result in the same performance given the train data. But, since we are mostly interested in future unseen data, I would choose the 3rd degree polynomial, $M = 3$, to avoid any over-fitting that occurs with higher degree polynomial fitting which would negatively affect the performance of the model in unforeseen data.
(Note that the $M = 4$ polynomial is slightly overfitted to the training data, since the MSE to more than 3 significant figures is slightly lower than that of $M = 3$. This is the effect we want to avoid, which becomes more pronounced given more training data, or on testing data.)

*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 $\alpha$. Note that in this example, we are using the same width $\alpha$ 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 $\alpha$.




As can be seen in the plot above, the best fit to the input data results from using radial basis functions with $\alpha = 100$. Since $\alpha$ represents the width of the Radial Basis Function, it can be deduced that too small values of $\alpha$ result in too narrow RBFs which only capture the points very close to their respective centers and produce a similar result to over-fitting (to those centers). On the opposite, too large values of $\alpha$ mean that the RBFs are too wide and basically capture all data points at once, and produce a similar result to under-fitting.

# Question 3 : (26 total points) Decision Trees

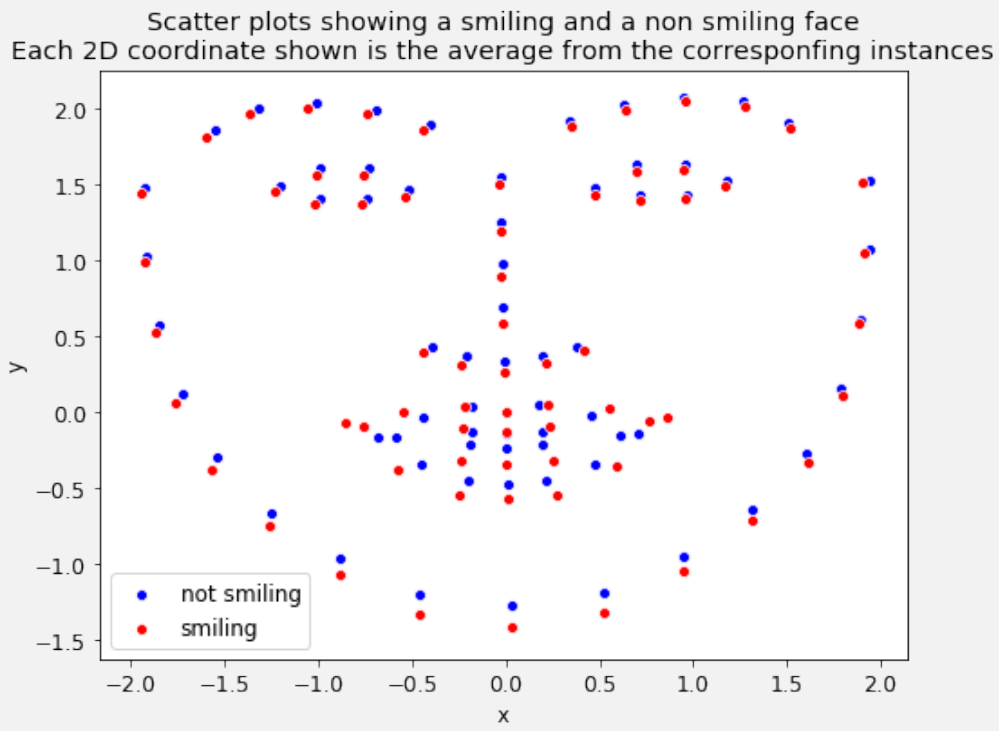
**In this question we will train a classifier to predict if a person is smiling or not.**

(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 the training and test splits contain as attributes 68 pairs of 2D coordinates $(x_i, y_i)$, all of type $float64$. The training set contains 4800 entries while the test set contains 1200. The target binary class, *smiling*, represents whether the specific instance of a face was smiling or not. Again, the training set contains 4800 target instances and the test set contains 1200 target instances.

*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 differences of the two face representations above are that the lips of the smiling face are wider, which makes sense because of the smile, and the chin of the smiling face is slightly lower than the chin of the non-smiling one.
Other parts of the faces, like the nose, eyes and eyebrows, also do not coincide perfectly, but the major differences are the ones mentioned above.

*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?

The DecisionTreeClassifier in sklearn uses the Gini impurity to measure the quality of a split. I believe both measures are equally accurate and there is no unique advantage of using this measure compared to entropy, besides the fact that the equation for entropy contains the slightly more computationally intensive *log*.

*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.

Smaller maximum depth means that the final Decision Tree has less leaf nodes to use for Classification. A potential problem of too small trees is that the model could be under-fitted to the data and thus may not be able to capture the class boundaries correctly.
On the other hand, as you increase the maximum depth, the leaf nodes increase, thus the tree becomes a better classifier on the training data, and can even reach 100% accuracy on training data if it ends up with all pure leaf nodes. But this means that the Decision tree is over-fitted to the training data and will perform worse on new data. In addition, if you have lots of attributes, the resulting tree will be very complicated, hard to understand, and computationally intensive to produce.

*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.*

| Maximum Depth | Train Accuracy | Test Accuracy |
|---|---|---|
| 2 | 79.48% | 78.17% |
| 8 | 93.35% | 84.08% |
| 20 | 100.00% | 81.58% |

The best model in this case is the decision tree classifier with maximum depth of 8, because it has the highest testing accuracy, which means it performed better than the other two in the new unseen data. The tree with maximum depth of 20 is over-fitted to the training data, which means that it can predict that data with 100% accuracy but does not perform as good to new unseen data.

*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 three most important attributes, in order of importance according to the Gini importance are:

1. $x_{50}$ with feature importance 0.330
2. $y_{48}$ with feature importance 0.090
3. $y_{29}$ with feature importance 0.088

The one with the highest importance represents the coordinate pair on the upper lip of the face. Thus it makes sense to be the most important, since the best way to distinguish a smiling face from a non-smiling one are the lips.

*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.

A limitation of this method is that for example a tilted head or a head with different size would produce different coordinates, thus the classifier would not be able to identify if the person is smiling or not. This is because in our case all the faces were pretty much similar in size and orientation, thus the difference of a smile was easily detectable.

# 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?

| Model | Classification Accuracy |
|---|---|
| alg_1 | 61.60% |
| alg_2 | 55.00% |
| alg_3 | 32.10% |
| alg_4 | 32.90% |

According to this metric, $alg\_1$ is the best since it has the highest classification accuracy. The limitation of the above method are that it cannot handle unbalanced classes. A way to improve it is to also include the Precision and Recall for each model, or the Miss and False alarm rates.

*Question 4 continued on next page...*

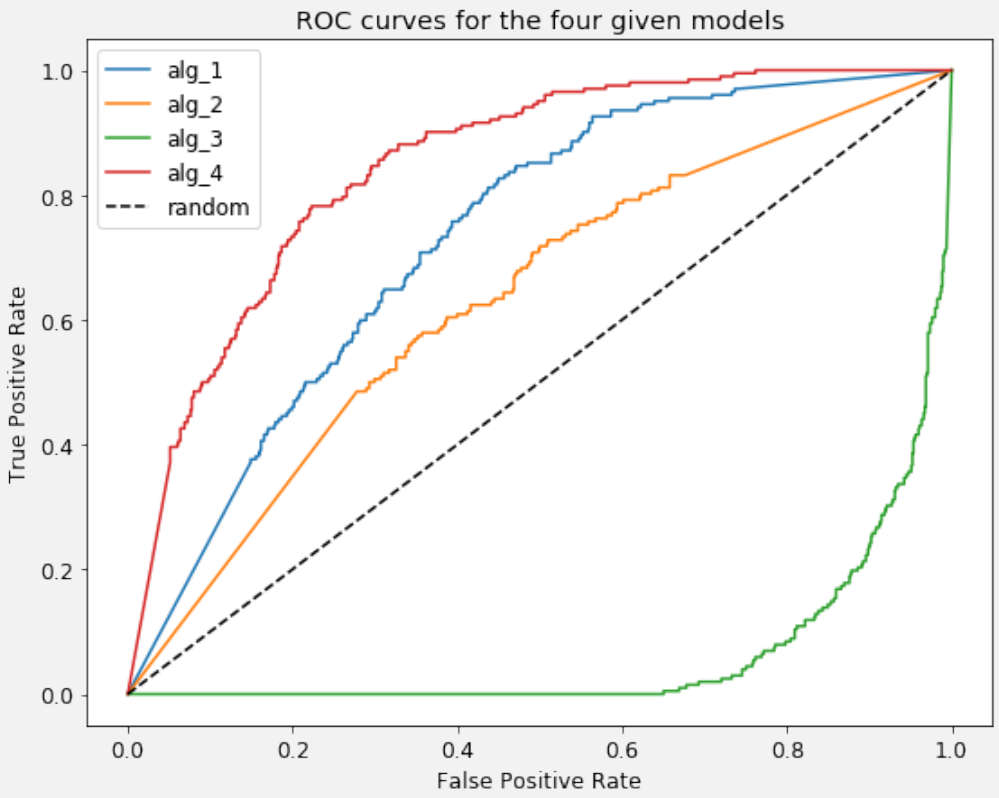
(b) (4 points) Instead of using classification accuracy, report the Area Under the ROC Curve (AUC) for each model. Does the model with the best AUC also have the best accuracy? If not, why not?
*Hint: You can use the roc_auc_score function from sklearn.*

| Model | AUC |
|---|---|
| alg_1 | 0.732 |
| alg_2 | 0.632 |
| alg_3 | 0.064 |
| alg_4 | 0.847 |

The model with the best AUC, which is $alg\_4$, does not have the best accuracy from part $(a)$. This is because the ROC curves are plotted using all possible threshold values, while in part $(a)$ the accuracy was calculated using only the threshold value of 0.5, which seems to favor $alg\_1$.

*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.*




As can be seen from the plot above, $alg\_3$ performs even worse than taking random guesses. A way to improve its performance without having to retrain the model is to swap the positive with the negative predictions of $alg\_3$, thus mirroring its ROC curve on the dashed line (the random classifier).